10-06-00

PATENT

A

Box PATENT APPLICATION
ASSISTANT COMMISSIONER FOR PATENTS
Washington, D.C. 20231

Date: October 4, 2000

Docket No. 30013630-0003

Sir:

Transmitted herewith for filing is the patent application of

Inventor(s): Peter Coad
Dietrich Charisius

For: Method and System for Displaying Changes of Source Code

I hereby certify that this paper is being deposited with the United States Postal Service as Express Mail in an envelope addressed to: Assistant Commissioner For Patents, Washington, D.C. 20231, on this date.

4 Oct 2000 — Marina N. Saito
Date

Express Mail Label No. EL676985638US

Enclosed are:

- ☒ 26 pages of specification, 8 pages of claims and an abstract.
- ☐ an executed oath or declaration, with power of attorney.
- ☒ an unexecuted oath or declaration, with power of attorney.
- ☒ 24 sheet(s) of informal drawing(s).
- ☐ _ sheet(s) of formal drawings(s).
- ☐ Assignment(s) of the invention to _____ .
- ☐ Assignment Form Cover Sheet.
- ☐ A check in the amount of $_____ to cover the fee for recording the assignment(s) is enclosed.
- ☐ Associate power of attorney.

Fee Calculation For Claims As Filed

a) Basic Fee						$	710.00
b) Independent Claims	8 - 3 = 5	X	$80.00	=		$	400.00
c) Total Claims	40 - 20 = 20	X	$18.00	=		$	360.00
d) Fee for Multiple Claims		X	$260.00	=		$	
			Total Filing Fee			$	1470.00

- ☒ Statement(s) of Status as Small Entity, reducing Filing Fee by half to $ 735.00
- ☐ Check No. in the amount of $ to cover the filing fee is enclosed
- ☐ Charge $_____ to Deposit Account No. 19-3140.
- ☐ Other _____.
- ☐ The Commissioner is hereby authorized to charge any additional fees which may be required to this application under 37 C.F.R. §§1.16-1.17, or credit any overpayment, to Deposit Account No. 19-3140. Should no proper payment be enclosed herewith, as by a check being in the wrong amount, unsigned, post-dated, otherwise improper or informal or even entirely missing, the Commissioner is authorized to charge the unpaid amount to Deposit Account No. 19-3140. Two duplicate copies of this sheet are enclosed.

SONNENSCHEIN NATH & ROSENTHAL
P. O. Box #061080
Wacker Drive Station, Sears Tower
Chicago, Illinois 60606-1080
(312) 876-8000

By: _____
Marina N. Saito
Registration No. 42,121

jc929 U.S. PTO 10/04/00
jc916 U.S. PTO 09/680065 10/04/00
Express Mail Label No. EL676985638US

Applicant or Patentee: Peter Coad; Dietrich Charisius; and Alexander Aptus
Attorney Docket No: 30013630-0003
Serial or Patent No.:
Filed or Issued:
For: Method And System For Displaying Changes Of Source Code

VERIFIED STATEMENT (DECLARATION) CLAIMING SMALL ENTITY STATUS
(37 CFR 1.9(f) AND 1.27(c) - SMALL BUSINESS CONCERN)

I hereby declare that I am
☐ the owner of the small business concern identified below:
☒ an official of the small business concern empowered to act on behalf of the concern identified below:

NAME OF CONCERN: TogetherSoft Corporation
ADDRESS OF CONCERN: 920 Main Campus Drive, Suite 410, Raleigh, North Carolina 27606

I hereby declare that the above identified small business concern qualifies as a small business concern as defined in 13 CFR 121.12, and reproduced in 37 CFR 1.9(d), for purposes of paying reduced fees to the United States Patent and Trademark Office, in that the number of employees of the concern, including those of its affiliates, does not exceed 500 persons. For purposes of this statement, (1) the number of employees of the business concern is the average over the previous fiscal year of the concern of the persons employed on a full-time, part-time, or temporary basis during each of the pay periods of the fiscal year, and (2) concerns are affiliates of each other when either, directly or indirectly, one concern controls or has the power to control the other, or a third party or parties controls or has the power to control both.

I hereby declare that rights under contract or law have been conveyed to and remain with the small business concern identified above with regard to the invention, entitled:
Method And System For Displaying Changes Of Source Code
By inventors Peter Coad; Dietrich Charisius; and Alexander Aptus

described in:
x the specification filed herewith.
Application Serial No.: _____, filed _____
Patent No.: _____, issued _____

If the rights held by the above-identified small business concern are not exclusive, each individual, concern, or organization having rights to the invention is listed below* and no rights to the invention are held by any person, other than the inventor, who would not qualify as an independent inventor under 37 CFR 1.9(c) if that person made the invention, or by any concern which would not qualify as a small business concern under 37 CFR 1.9(d), or a nonprofit organization under 37 CFR 1.9(e).
*NOTE: Separate verified statements are required from each named person, concern, or organization having rights to the invention averring to their status as small entities. (37 CFR 1.27)

FULL NAME _____
ADDRESS _____
☐ INDIVIDUAL SMALL BUSINESS CONCERN ☐ NONPROFIT ORGANIZATION

I acknowledge the duty to file, in this application or patent, notification of any change in status resulting in loss of entitlement to small entity status prior to paying, or at the time of paying, the earliest of the issue fee or any maintenance fee due after the date on which status as a small entity is no longer appropriate. (37 CFR 1.28(b))

I hereby declare that all statements made herein of my own knowledge are true, and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application, any patent issuing thereon, or any patent to which this verified statement is directed.

NAME OF PERSON SIGNING: Steve Morrison
TITLE OF PERSON OTHER THAN OWNER: Vice President
ADDRESS OF PERSON SIGNING: 920 Main Campus Drive, Suite 410, Raleigh, NC 27606
SIGNATURE: _____ DATE: 10/3/00

UNITED STATES PATENT APPLICATION

OF

PETER COAD

AND

DIETRICH CHARISIUS

FOR

METHOD AND SYSTEM FOR DISPLAYING CHANGES OF SOURCE CODE

04470.P5050

METHOD AND SYSTEM FOR DISPLAYING CHANGES OF SOURCE CODE

Cross-Reference To Related Applications

The following identified U.S. patent applications are relied upon and are incorporated by reference in this application:

U.S. Provisional Application No. 60/157,826, entitled "Visual Unified Modeling Language Development Tool," filed on October 5, 1999;

U.S. Provisional Application No. 60/199,046, entitled "Software Development Tool," filed on April 21, 2000;

U.S. Patent Application No. _____, entitled "Method And System For Developing Software," bearing attorney docket no. 30013630-0002, and filed on the same date herewith;

U.S. Patent Application No. _____, entitled "Method And System For Generating, Applying, And Defining A Pattern," bearing attorney docket no. 30013630-0004, and filed on the same date herewith; and

U.S. Patent Application No. _____, entitled "Method And System For Collapsing A Graphical Representation Of Related Elements," bearing attorney docket no. 30013630-0005, and filed on the same date herewith.

Field Of The Invention

The present invention relates to a method and system for developing software. More particularly, the invention relates to a method and system for tracking changes made to source code, and displaying the source code with these changes.

Background Of The Invention

Computer instructions are written in source code. Although a skilled programmer can understand source code to determine what the code is designed to accomplish, with highly complex software systems, a graphical representation or model of the source code is helpful to organize and visualize the structure and components of the system. Using models, the complex systems are easily identified, and the structural and behavioral patterns can be visualized and documented.

The well-known Unified Modeling Language (UML) is a general-purpose notational language for visualizing, specifying, constructing, and documenting complex software systems. UML is used to model systems ranging from business information systems to Web-based distributed systems, to real-time embedded systems. UML formalizes the notion that real-world objects are best modeled as self-contained entities that contain both data and functionality. UML is more clearly described in the following references, which are incorporated herein by reference: (1) Martin Fowler, UML Distilled Second Edition: Applying the Standard Object Modeling Language, Addison-Wesley (1999); (2) Booch, Rumbaugh, and Jacobson, The Unified Modeling Language User Guide, Addison-Wesley (1998); (3) Peter Coad, Jeff DeLuca, and Eric Lefebvre, Java Modeling in Color with UML: Enterprise Components and Process, Prentice Hall (1999); and (4) Peter Coad, Mark Mayfield, and Jonathan Kern, Java Design: Building Better Apps & Applets (2nd Ed.), Prentice Hall (1998).

As shown in Fig. 1, conventional software development tools 100 allow a programmer to view UML 102 while viewing source code 104. The source code 104 is stored in a file, and a reverse engineering module 106 converts the source code 104 into a representation of the software project in a database or repository 108. The software project comprises source code 104 in at least one file which, when compiled, forms a sequence of instructions to be run by the data processing system. The repository 108 generates the UML 102. If any changes are made to the UML 102, they are automatically reflected in the repository 108, and a code generator 110 converts the representation in the repository 108 into source code 104. Such software development tools 100, however, do not synchronize the displays of the UML 102 and the source code 104. Rather, the repository 108 stores the representation of the software project while the file stores the source code 104. A modification in the UML 102 does not appear in the source code 104 unless the code generator 110 re-generates the source code 104 from the data in the repository 108. When this occurs, the entire source code 104 is rewritten. Similarly, any modifications made to the source code 104 do not appear in the UML 102 unless the reverse engineering module 106 updates the repository 108. As a result, redundant information is stored in the repository 108 and the source code 104. In addition, rather than making incremental changes to the source code 104, conventional software development tools 100 rewrite the overall source code 104 when modifications are made to the UML 102, resulting in wasted

processing time. This type of manual, large-grained synchronization requires either human intervention, or a "batch" style process to try to keep the two views (the UML 102 and the source code 104) in sync. Unfortunately, this approach, adopted by many tools, leads to many undesirable side-effects; such as desired changes to the source code being overwritten by the tool. A further disadvantage with conventional software development tools 100 is that they are designed to only work in a single programming language. Thus, a tool 100 that is designed for Java™ programs cannot be utilized to develop a program in C++. Moreover, it is not possible to track the changes made to the source code using conventional software development tools. Accordingly, unless the developer maintains detailed notes regarding the development of the source code, it is difficult to determine what modifications to the source code were attempted during its development. There is a need in the art for a tool that avoids the limitations of these conventional software development tools.

Summary Of The Invention

Methods and systems consistent with the present invention provide an improved software development tool which overcomes the limitations of conventional software development tools. The improved software development tool of the present invention allows a developer to track changes made to source code, and display the source code with these changes. Accordingly, the developer can recollect the modifications that were made to the source code during the development of the source code.

In accordance with methods consistent with the present invention, a method is provided in a data processing system for displaying versions of source code. Each version reflects an instance in an edit history. The method comprises the steps of storing indications of the edits to the source code, and displaying the versions of the source code with the indications of the edits.

In accordance with articles of manufacture consistent with the present invention, a computer-readable medium is provided. The computer-readable medium contains instructions for controlling a data processing system to perform a method. The data processing system has versions of source code. Each version reflects an instance in an edit history. The method comprises the steps of storing indications of the edits to the source code, and displaying the versions of the source code with the indications of the edits.

Other systems, methods, features and advantages of the invention will be or will become apparent to one with skill in the art upon examination of the following figures and detailed description. It is intended that all such additional systems, methods, features and advantages be included within this description, be within the scope of the invention, and be protected by the accompanying claims.

Brief Description Of The Drawings

The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate an implementation of the invention and, together with the description, serve to explain the advantages and principles of the invention. In the drawings,

Fig. 1 depicts a conventional software development tool;

Fig. 2 depicts an overview of a software development tool in accordance with the present invention;

Fig. 3 depicts a data structure of the language-neutral representation created by the software development tool of Fig. 2;

Fig. 4 depicts representative source code;

Fig. 5 depicts the data structure of the language-neutral representation of the source code of Fig. 4;

Fig. 6 depicts a data processing system suitable for practicing the present invention;

Fig. 7 depicts an architectural overview of the software development tool of Fig. 2;

Fig. 8A depicts a user interface displayed by the software development tool depicted in Fig. 2, where the user interface displays a list of predefined criteria which the software development tool checks in the source code;

Fig. 8B depicts a user interface displayed by the software development tool depicted in Fig. 2, where the user interface displays the definition of the criteria which the software development tool checks in the source code, and an example of source code which does not conform to the criteria;

Fig. 8C depicts a user interface displayed by the software development tool depicted in Fig. 2, where the user interface displays an example of source code which conforms to the criteria which the software development tool checks in the source code;

Fig. 9 depicts a flow diagram of the steps performed by the software development tool depicted in Fig. 2;

Figs. 10A and 10B depict a flow diagram illustrating the update model step of Fig. 9;

Fig. 11 depicts a flow diagram of the steps performed by the software development tool in Fig. 2 when creating a class;

Fig. 12 depicts a user interface displayed by the software development tool depicted in Fig. 2, where the user interface displays a use case diagram of source code;

Fig. 13 depicts a user interface displayed by the software development tool depicted in Fig. 2, where the user interface displays both a class diagram and a textual view of source code;

Fig. 14 depicts a user interface displayed by the software development tool depicted in Fig. 2, where the user interface displays a sequence diagram of source code;

Fig. 15 depicts a user interface displayed by the software development tool depicted in Fig. 2, where the user interface displays a collaboration diagram of source code;

Fig. 16 depicts a user interface displayed by the software development tool depicted in Fig. 2, where the user interface displays a statechart diagram of source code;

Fig. 17 depicts a user interface displayed by the software development tool depicted in Fig. 2, where the user interface displays an activity diagram of source code;

Fig. 18 depicts a user interface displayed by the software development tool depicted in Fig. 2, where the user interface displays a component diagram of source code;

Fig. 19 depicts a user interface displayed by the software development tool depicted in Fig. 2, where the user interface displays a deployment diagram of source code;

Fig. 20 depicts a flow diagram of the steps performed by the software development tool depicted in Fig. 2 to store the edit history of source code, in accordance with the present invention; and

Figs. 21A-C depict a flow diagram of the steps performed by the software development tool depicted in Fig. 2 to sequentially displaying the source code with the indications of the edit history, in accordance with the present invention.

Detailed Description Of The Invention

Methods and systems consistent with the present invention provide an improved software development tool which displays versions of source code. Each version reflects an instance in an edit history, i.e., reflects the changes made to the source code.

As depicted in Fig. 2, source code 202 is being displayed in both a graphical form 204 and a textual form 206. In accordance with methods and systems consistent with the present invention, the improved software development tool generates a transient meta model (TMM) 200 which stores a language-neutral representation of the source code 202. The graphical 204 and textual 206 representations of the source code 202 are generated from the language-neutral representation in the TMM 200. Although modifications made on the displays 204 and 206 may appear to modify the displays 204 and 206, in actuality all modifications are made directly to the source code 202 via an incremental code editor (ICE) 208, and the TMM 200 is used to generate the modifications in both the graphical 204 and the textual 206 views from the modifications to the source code 202.

The improved software development tool provides simultaneous round-trip engineering, i.e., the graphical representation 204 is synchronized with the textual representation 206. Thus, if a change is made to the source code 202 via the graphical representation 204, the textual representation 206 is updated automatically. Similarly, if a change is made to the source code 202 via the textual representation 206, the graphical representation 204 is updated to remain synchronized. There is no repository, no batch code generation, and no risk of losing code.

The data structure 300 of the language-neutral representation is depicted in Fig. 3. The data structure 300 comprises a Source Code Interface (SCI) model 302, an SCI package 304, an SCI class 306, and an SCI member 308. The SCI model 302 is the source code organized into packages. The SCI model 302 corresponds to a directory for a software project being developed by the user, and the SCI package 304 corresponds to a subdirectory. The software project comprises the source code in at least one file that is compiled to form a sequence of instructions to be run by a data processing system. The data processing system is discussed in detail below. As is well known in object-oriented programming, the class 306 is a category of objects which describes a group of objects with similar properties (attributes), common behavior (operations or methods), common

relationships to other objects, and common semantics. The members 308 comprise attributes and/or operations.

For example, the data structure 500 for the source code 400 depicted in Fig. 4 is depicted in Fig. 5. UserInterface 402 is defined as a package 404. Accordingly, UserInterface 402 is contained in SCI package 502. Similarly, Bank 406, which is defined as a class 408, is contained in SCI class 504, and Name 410 and Assets 412, which are defined as attributes (strings 414), are contained in SCI members 506. Since these elements are in the same project, all are linked. The data structure 500 also identifies the language in which the source code is written 508, e.g., the Java™ language.

Fig. 6 depicts a data processing system 600 suitable for practicing methods and systems consistent with the present invention. Data processing system 600 comprises a memory 602, a secondary storage device 604, an I/O device 606, and a processor 608. Memory 602 includes the improved software development tool 610. The software development tool 610 is used to develop a software project 612, and create the TMM 200 in the memory 602. The project 612 is stored in the secondary storage device 604 of the data processing system 600. One skilled in the art will recognize that data processing system 600 may contain additional or different components.

Although aspects of the present invention are described as being stored in memory, one skilled in the art will appreciate that these aspects can also be stored on or read from other types of computer-readable media, such as secondary storage devices, like hard disks, floppy disks or CD-ROM; a carrier wave from a network, such as Internet; or other forms of RAM or ROM either currently known or later developed.

Fig. 7 illustrates an architectural overview of the improved software development tool 610. The tool 610 comprises a core 700, an open application program interface (API) 702, and modules 704. The core 700 includes a parser 706 and an ICE 208. The parser 706 converts the source code into the language-neutral representation in the TMM, and the ICE 208 converts the text from the displays into source code. There are three main packages composing the API 702: Integrated Development Environment (IDE) 708; Read-Write Interface (RWI) 710; and Source Code Interface (SCI) 712. Each package includes corresponding subpackages. As is well known in the art, a package is a collection of attributes, notifications, operations, or behaviors that are treated as a single module or program unit.

IDE 708 is the API 702 needed to generate custom outputs based on information contained in a model. It is a read-only interface, i.e., the user can extract information from the model, but not change the model. IDE 708 provides the functionality related to the model's representation in IDE 708 and interaction with the user. Each package composing the IDE group has a description highlighting the areas of applicability of this concrete package.

RWI 710 enables the user to go deeper into the architecture. Using RWI 710, information can be extracted from and written to the models. RWI not only represents packages, classes and members, but it may also represent different diagrams (class diagrams, use case diagrams, sequence diagrams and others), links, notes, use cases, actors, states, etc.

SCI 712 is at the source code level, and allows the user to work with the source code almost independently of the language being used.

There are a variety of modules 704 in the software development tool 610 of the present invention. Some of the modules 704 access information to generate graphical and code documentation in custom formats, export to different file formats, or develop patterns. The software development tool also includes a quality assurance (QA) module which monitors the modifications to the source code and calculates the complexity metrics, i.e., the measurement of the program's performance or efficiency, to support quality assurance. The types of metrics calculated by the software development tool include basic metrics, cohesion metrics, complexity metrics, coupling metrics, Halstead metrics, inheritance metrics, maximum metrics, polymorphism metrics, and ratio metrics. Examples of these metrics with their respective definitions are identified in Tables 1-9 below.

Basic Metrics	Description
Lines Of Code	Counts the number of code lines.
Number Of Attributes	Counts the number of attributes. If a class has a high number of attributes, it may be appropriate to divide it into subclasses.
Number Of Classes	Counts the number of classes.
Number Of Import Statements	Counts the number of imported packages/classes. This measure can highlight excessive importing, and also can be used as a measure of coupling.
Number Of Members	Counts the number of members, i.e., attributes and operations. If a class has a high number of members, it may be appropriate to divide it into subclasses.
Number Of Operations	Counts the number of operations. If a class has a high number of operations, it may be appropriate to divide it into subclasses.

Table 1 – Basic Metrics

Cohesion Metrics	Description
Lack Of Cohesion Of Methods 1	Takes each pair of methods in the class and determines the set of fields they each access. A low value indicates high coupling between methods, which indicates potentially low reusability and increased testing because many methods can affect the same attributes.
Lack Of Cohesion Of Methods 2	Counts the percentage of methods that do not access a specific attribute averaged over all attributes in the class. A high value of cohesion (a low lack of cohesion) implies that the class is well designed.
Lack Of Cohesion Of Methods 3	Measures the dissimilarity of methods in a class by attributes. A low value indicates good class subdivision, implying simplicity and high reusability. A high lack of cohesion increases complexity, thereby increasing the likelihood of errors during the development process.

Table 2 – Cohesion Metrics

Complexity Metrics	Description
Attribute Complexity	Defined as the sum of each attribute's value in the class.
Cyclomatic Complexity	Represents the cognitive complexity of the class. It counts the number of possible paths through an algorithm by counting the number of distinct regions on a flowgraph, i.e., the number of 'if,' 'for' and 'while' statements in the operation's body.
Number Of Remote Methods	Processes all of the methods and constructors, and counts the number of different remote methods called. A remote method is defined as a method which is not declared in either the class itself or its ancestors.
Response For Class	Calculated as 'Number of Local Methods' + 'Number of Remote Methods.' A class which provides a larger response set is considered to be more complex and requires more testing than one with a smaller overall design complexity.
Weighted Methods Per Class 1	The sum of the complexity of all methods for a class, where each method is weighted by its cyclomatic complexity. The number of methods and the complexity of the methods involved is a predictor of how much time and effort is required to develop and maintain the class.
Weighted Methods Per Class 2	Measures the complexity of a class, assuming that a class with more methods than another is more complex, and that a method with more parameters than another is also likely to be more complex.

Table 3 – Complexity Metrics

Coupling Metrics	Description
Coupling Between Objects	Represents the number of other classes to which a class is coupled. Counts the number of reference types that are used in attribute declarations, formal parameters, return types, throws declarations and local variables, and types from which attribute and method selections are made. Excessive coupling between objects is detrimental to modular design and prevents reuse. The more independent a class is, the easier it is to reuse it in another application. In order to improve modularity and promote encapsulation, inter-object class couples should be kept to a minimum. The larger the number of couples, the higher the sensitivity to changes in other parts of the design, and therefore maintenance is more difficult. A measure of coupling is useful to determine how complex the testing of various parts of a design is likely to be. The higher the inter-object class coupling, the more rigorous the testing needs to be.
Data Abstraction Coupling	Counts the number of reference types used in the attribute declarations.
FanOut	Counts the number of reference types that are used in attribute declarations, formal parameters, return types, throws declarations and local variables.

Table 4 – Coupling Metrics

Halstead Metrics	Description
Halstead Difficulty	This measure is one of the Halstead Software Science metrics. It is calculated as ('Number of Unique Operators' / 'Number of Unique Operands') * ('Number of Operands' / 'Number of Unique Operands').
Halstead Effort	This measure is one of the Halstead Software Science metrics. It is calculated as 'Halstead Difficulty' * 'Halstead Program Volume.'
Halstead Program Length	This measure is one of the Halstead Software Science metrics. It is calculated as 'Number of Operators' + 'Number of Operands.'
Halstead Program Vocabulary	This measure is one of the Halstead Software Science metrics. It is calculated as 'Number of Unique Operators' + 'Number of Unique Operands.'
Halstead Program Volume	This measure is one of the Halstead Software Science metrics. It is calculated as 'Halstead Program Length' * Log2('Halstead Program Vocabulary').
Number Of Operands	This measure is used as an input to the Halstead Software Science metrics. It counts the number of operands used in a class.
Number Of Operators	This measure is used as an input to the Halstead Software Science metrics. It counts the number of operators used in a class.
Number Of Unique Operands	This measure is used as an input to the Halstead Software Science metrics. It counts the number of unique operands used in a class.
Number Of Unique Operators	This measure is used as an input to the Halstead Software Science metrics. It counts the number of unique operators used in a class.

Table 5 – Halstead Metrics

Inheritance Metrics	Description
Depth Of Inheritance Hierarchy	Counts how far down the inheritance hierarchy a class or interface is declared. High values imply that a class is quite specialized.
Number Of Child Classes	Counts the number of classes which inherit from a particular class, i.e., the number of classes in the inheritance tree down from a class. Non-zero value indicates that the particular class is being re-used. The abstraction of the class may be poor if there are too many child classes. It should also be stated that a high value of this measure points to the definite amount of testing required for each child class.

Table 6 – Inheritance Metrics

Maximum Metrics	Description
Maximum Number Of Levels	Counts the maximum depth of 'if,' 'for' and 'while' branches in the bodies of methods. Logical units with a large number of nested levels may need implementation simplification and process improvement because groups that contain more than seven pieces of information are increasingly harder for people to understand in problem solving.
Maximum Number Of Parameters	Displays the maximum number of parameters among all class operations. Methods with many parameters tend to be more specialized and, thus, are less likely to be reusable.
Maximum Size Of Operation	Counts the maximum size of the operations for a class. Method size is determined in terms of cyclomatic complexity, i.e., the number of 'if,' 'for' and 'while' statements in the operation's body.

Table 7 – Maximum Metrics

Polymorphism Metrics	Description
Number Of Added Methods	Counts the number of operations added by a class. A large value of this measure indicates that the functionality of the given class becomes increasingly distinct from that of the parent classes. In this case, it should be considered whether this class genuinely should be inheriting from the parent, or if it could be broken down into several smaller classes.
Number Of Overridden Methods	Counts the number of inherited operations which a class overrides. Classes without parents are not processed. High values tend to indicate design problems, i.e., subclasses should generally add to and extend the functionality of the parent classes rather than overriding them.

Table 8 – Polymorphism Metrics

Ratio Metrics	Description
Comment Ratio	Counts the ratio of comments to total lines of code including comments.
Percentage Of Package Members	Counts the percentage of package members in a class.
Percentage Of Private Members	Counts the percentage of private members in a class.
Percentage Of Protected Members	Counts the percentage of protected members in a class.
Percentage Of Public Members	Counts the proportion of vulnerable members in a class. A large proportion of such members means that the class has high potential to be affected by external classes and means that increased efforts will be needed to test such a class thoroughly.
True Comment Ratio	Counts the ratio of comments to total lines of code excluding comments.

Table 9 – Ratio Metrics

The QA module also provides audits, i.e., the module checks for conformance to predefined or user-defined styles. The types of audits provided by the module include coding style, critical errors, declaration style, documentation, naming style, performance, possible errors and superfluous content. Examples of these audits with their respective definitions are identified in Tables 10-17 below.

Coding Style Audits	Description
Access Of Static Members Through Objects	Static members should be referenced through class names rather than through objects.
Assignment To Formal Parameters	Formal parameters should not be assigned.
Complex Assignment	Checks for the occurrence of multiple assignments and assignments to variables within the same expression. Complex assignments should be avoided since they decrease program readability.
Don't Use the Negation Operator Frequently	The negation operator slows down the readability of the program. Thus, it is recommended that it not be used frequently.
Operator '?:' May Not Be Used	The operator '?:' makes the code harder to read than the alternative form with an if-statement.
Provide Incremental In For-Statement or use while-statement	Checks if the third argument of the 'for'-statement is missing.
Replacement For Demand Imports	Demand import-declarations must be replaced by a list of single import-declarations that are actually imported into the compilation unit. In other words, import-statements may not end with an asterisk.
Use Abbreviated Assignment Operator	Use the abbreviated assignment operator in order to write programs more rapidly. Also some compilers run faster with the abbreviated assignment operator.
Use 'this' Explicitly To Access Class Members	Tries to make the developer use 'this' explicitly when trying to access class members. Using the same class member names with parameter names often makes what the developer is referring to unclear.

Table 10 – Coding Style Audits

Critical Errors Audits	Description
Avoid Hiding Inherited Attributes	Detects when attributes declared in child classes hide inherited attributes.
Avoid Hiding Inherited Static Methods	Detects when inherited static operations are hidden by child classes.
Command Query Separation	Prevents methods that return a value from a modifying state. The methods used to query the state of an object must be different from the methods used to perform commands (change the state of the object).
Hiding Of Names	Declarations of names should not hide other declarations of the same name.
Inaccessible Constructor Or Method Matches	Overload resolution only considers constructors and methods that are visible at the point of the call. If, however, all the constructors and methods were considered, there may be more matches. This rule is violated in this case. Imagine that ClassB is in a different package than ClassA. Then the allocation of ClassB violates this rule since the second constructor is not visible at the point of the allocation, but it still matches the allocation (based on signature). Also the call to open in ClassB violates this rule since the second and the third declarations of open are not visible at the point of the call, but it still matches the call (based on signature).
Multiple Visible Declarations With Same Name	Multiple declarations with the same name must not be simultaneously visible except for overloaded methods.
Overriding a Non-Abstract Method With an Abstract Method	Checks for abstract methods overriding non-abstract methods in a subclass.
Overriding a Private Method	A subclass should not contain a method with the same name and signature as in a superclass if these methods are declared to be private.
Overloading Within a Subclass	A superclass method may not be overloaded within a subclass unless all overloading in the superclass are also overridden in the subclass. It is very unusual for a subclass to be overloading methods in its superclass without also overriding the methods it is overloading. More frequently this happens due to inconsistent changes between the superclass and subclass – i.e., the intention of the user is to override the method in the superclass, but due to the error, the subclass method ends up overloading the superclass method.
Use of Static Attribute for Initialization	Non-final static attributes should not be used in initializations of attributes.

Table 11 – Critical Errors Audits

Declaration Style Audits	Description
Badly Located Array Declarators	Array declarators must be placed next to the type descriptor of their component type.
Constant Private Attributes Must Be Final	Private attributes that never get their values changed must be declared final. By explicitly declaring them in such a way, a reader of the source code get some information of how the attribute is supposed to be used.
Constant Variables Must Be Final	Local variables that never get their values changed must be declared final. By explicitly declaring them in such a way, a reader of the source code obtains information about how the variable is supposed to be used.
Declare Variables In One Statement Each	Several variables (attributes and local variables) should not be declared in the same statement.
Instantiated Classes Should Be Final	This rule recommends making all instantiated classes final. It checks classes which are present in the object model. Classes from search/classpath are ignored.
List All Public And Package Members First	Enforces a standard to improve readability. Methods/data in your class should be ordered properly.
Order Of Appearance Of Modifiers	Checks for correct ordering of modifiers. For classes, this includes visibility (public, protected or private), abstract, static, final. For attributes, this includes visibility (public, protected or private), static, final, transient, volatile. For operations, this includes visibility (public, protected or private), abstract, static, final, synchronized, native.
Put the Main Function Last	Tries to make the program comply with various coding standards regarding the form of the class definitions.

Table 12 – Declaration Style Audits

Documentation Audits	Description
Bad Tag In JavaDoc Comments	This rule verifies code against accidental use of improper JavaDoc tags.
Distinguish Between JavaDoc And Ordinary Comments	Checks whether the JavaDoc comments in your program ends with '**/' and ordinary C-style ones with '*/.'

Table 13 – Documentation Audits

Naming Style Audits	Description
Class Name Must Match Its File Name	Checks whether top level classes or interfaces have the same name as the file in which they reside.
Group Operations With Same Name Together	Enforces standard to improve readability.
Naming Conventions	Takes a regular expression and item name and reports all occurrences where the pattern does not match the declaration.
Names Of Exception Classes	Names of classes which inherit from Exception should end with Exception.
Use Conventional Variable Names	One-character local variable or parameter names should be avoided, except for temporary and looping variables, or where a variable holds an undistinguished value of a type.

Table 14 – Naming Style Audits

Performance Audits	Description
Avoid Declaring Variables Inside Loops	This rule recommends declaring local variables outside the loops since declaring variables inside the loop is less efficient.
Append To String Within a Loop	Performance enhancements can be obtained by replacing String operations with StringBuffer operations if a String object is appended within a loop.
Complex Loop Expressions	Avoid using complex expressions as repeat conditions within loops.

Table 15 – Performance Audits

Possible Error Audits	Description
Avoid Public And Package Attributes	Declare the attributes either private or protected, and provide operations to access or change them.
Avoid Statements With Empty Body	Avoid statements with empty body.
Assignment To For-Loop Variables	'For'-loop variables should not be assigned.
Don't Compare Floating Point Types	Avoid testing for equality of floating point numbers since floating-point numbers that should be equal are not always equal due to rounding problems.
Enclosing Body Within a Block	The statement of a loop must always be a block. The 'then' and 'else' parts of 'if'-statements must always be blocks. This makes it easier to add statements without accidentally introducing bugs in case the developer forgets to add braces.
Explicitly Initialize All Variables	Explicitly initialize all variables. The only reason not to initialize a variable is where it's declared is if the initial value depends on some computation occurring first.
Method finalize() Doesn't Call super.finalize()	Calling of super.finalize() from finalize() is good practice of programming, even if the base class doesn't define the finalize() method. This makes class implementations less dependent on each other.
Mixing Logical Operators Without Parentheses	An expression containing multiple logical operators should be parenthesized properly.
No Assignments In Conditional Expressions	Use of assignment within conditions makes the source code hard to understand.
Use 'equals' Instead Of '=='	The '==' operator used on strings checks if two string objects are two identical objects. In most situations, however, one likes to simply check if two strings have the same value. In these cases, the 'equals' method should be used.
Use 'L' Instead Of 'l' at the end of integer constant	It is better to use uppercase 'L' to distinguish the letter 'l' from the number '1.'
Use Of the 'synchronized' Modifier	The 'synchronized' modifier on methods can sometimes cause confusion during maintenance as well as during debugging. This rule therefore recommends against using this modifier, and instead recommends using 'synchronized' statements as replacements.

Table 16 – Possible Error Audits

Superfluous Content Audits	Description
Duplicate Import Declarations	There should be at most one import declaration that imports a particular class/package.
Don't Import the Package the Source File Belongs To	No classes or interfaces need to be imported from the package to which the source code file belongs. Everything in that package is available without explicit import statements.
Explicit Import Of the java.lang Classes	Explicit import of classes from the package 'java.lang' should not be performed.
Equality Operations On Boolean Arguments	Avoid performing equality operations on Boolean operands. 'True' and 'false' literals should not be used in conditional clauses.
Imported Items Must Be Used	It is not legal to import a class or an interface and never use it. This rule checks classes and interfaces that are explicitly imported with their names – that is not with import of a complete package, using an asterisk. If unused class and interface imports are omitted, the amount of meaningless source code is reduced - thus the amount of code to be understood by a reader is minimized.
Unnecessary Casts	Checks for the use of type casts that are not necessary.
Unnecessary 'instanceof' Evaluations	Verifies that the runtime type of the left-hand side expression is the same as the one specified on the right-hand side.
Unused Local Variables And Formal Parameters	Local variables and formal parameter declarations must be used.
Use Of Obsolete Interface Modifier	The modifier 'abstract' is considered obsolete and should not be used.
Use Of Unnecessary Interface Member Modifiers	All interface operations are implicitly public and abstract. All interface attributes are implicitly public, final and static.
Unused Private Class Member	An unused class member might indicate a logical flaw in the program. The class declaration has to be reconsidered in order to determine the need of the unused member(s).

Table 17 – Superfluous Content Audits

If the QA module determines that the source code does not conform, an error message is provided to the developer. For example, as depicted in Fig. 8A, the software development tool checks for a variety of coding styles 800. If the software development tool were to check for "Access Of Static Members Through Objects" 802, it would verify whether static members are referenced through class names rather than through objects 804. Further, as depicted in Fig. 8B, if the software development tool were to check for

"Complex Assignment" 806, the software development tool would check for the occurrence of multiple assignments and assignments to variables within the same expression to avoid complex assignments since these decrease program readability 808. An example of source code having a complex assignment 810 and source code having a non-complex assignment 812 are depicted in Figs. 8B and 8C, respectively. The QA module of the software development tool monitors the source code for other syntax errors well known in the art, as described above, and provides an error message if any such errors are detected.

The improved software development tool of the present invention is used to develop source code in a project. The project comprises a plurality of files and the source code of one of the plurality of files is written in a given language. The software development tool determines the language of the source code of the file, converts the source code from the language into a language-neutral representation, uses the language-neutral representation to textually display the source code of the file in the language, and uses the language-neutral representation to display a graphical representation of at least a portion of the project. The source code and the graphical representation are displayed simultaneously.

The improved software development tool of the present invention is also used to develop source code. The software development tool receives an indication of a selected language for the source code, creates a file to store the source code in the selected language, converts the source code from the selected language into a language-neutral representation, uses the language-neutral representation to display the source code of the file, and uses the language-neutral representation to display a graphical representation of the file. Again, the source code and the graphical representation are displayed simultaneously.

Moreover, if the source code in the file is modified, the modified source code and a graphical representation of at least a portion of the modified source code are displayed simultaneously. The QA module of the software development tool provides an error message if the modification does not conform to predefined or user-defined styles, as described above. The modification to the source code may be received from the display of the source code, the display of the graphical representation of the project, or via some other independent software to modify the code. The graphical representation of the project may

be in Unified Modeling Language; however, one skilled in the art will recognize that other graphical representations of the source code may be displayed. Further, although the present invention is described and shown using the various views of the UML, one of ordinary skill in the art will recognize that other views may be displayed.

Fig. 9 depicts a flow diagram of the steps performed by the software development tool to develop a project in accordance with the present invention. As previously stated, the project comprises a plurality of files. The developer either uses the software development tool to open a file which contains existing source code, or to create a file in which the source code will be developed. If the software development tool is used to open the file, determined in step 900, the software development tool initially determines the programming language in which the code is written (step 902). The language is identified by the extension of the file, e.g., ".java" identifies source code written in the Java™ language, while ".cpp" identifies source code written in C++. The software development tool then obtains a template for the current programming language, i.e., a collection of generalized definitions for the particular language that can be used to build the data structure (step 904). For example, the definition of a new Java™ class contains a default name, e.g., "Class1," and the default code, "public class Class1 {}." Such templates are well known in the art. For example, the "Microsoft Foundation Class Library" and the "Microsoft Word Template For Business Use Case Modeling" are examples of standard template libraries from which programmers can choose individual template classes. The software development tool uses the template to parse the source code (step 906), and create the data structure (step 908). After creating the data structure or if there is no existing code, the software development tool awaits an event, i.e., a modification or addition to the source code by the developer (step 910). If an event is received and the event is to close the file (step 912), the file is saved (step 914) and closed (step 916). Otherwise, the software development tool performs the event (step 918), i.e., the tool makes the modification. The software development tool then updates the TMM or model (step 920), as discussed in detail below, and updates both the graphical and the textual views (step 922).

Figs. 10A and 10B depict a flow diagram illustrating the update model step of Fig. 9. The software development tool selects a file from the project (step 1000), and determines whether the file is new (step 1002), whether the file has been updated (step

1004), or whether the file has been deleted (step 1006). If the file is new, the software development tool adds the additional symbols from the file to the TMM (step 1008). To add the symbol to the TMM, the software development tool uses the template to parse the symbol to the TMM. If the file has been updated, the software development tool updates the symbols in the TMM (step 1010). Similar to the addition of a symbol to the TMM, the software development tool uses the template to parse the symbol to the TMM. If the file has been deleted, the software development tool deletes the symbols in the TMM (step 1012). The software development tool continues this analysis for all files in the project. After all files are analyzed (step 1014), any obsolete symbols in the TMM (step 1016) are deleted (step 1018).

Fig. 11 depicts a flow diagram illustrating the performance of an event, specifically the creation of a class, in accordance with the present invention. After identifying the programming language (step 1100), the software development tool obtains a template for the language (step 1102), creates a source code file in the project directory (step 1104), and pastes the template onto the TMM (step 1106). The project directory corresponds to the SCI model 302 of Fig. 3. Additional events which a developer may perform using the software development tool include the creation, modification or deletion of packages, projects, attributes, interfaces, links, operations, and the closing of a file.

The software development tool is collectively broken into three views of the application: the static view, the dynamic view, and the functional view. The static view is modeled using the use-case and class diagrams. A use case diagram 1200, depicted in Fig. 12, shows the relationship among actors 1202 and use cases 1204 within the system 1206. A class diagram 1300, depicted in Fig. 13 with its associated source code 1302, on the other hand, includes classes 1304, interfaces, packages and their relationships connected as a graph to each other and to their contents.

The dynamic view is modeled using the sequence, collaboration and statechart diagrams. As depicted in Fig. 14, a sequence diagram 1400 represents an interaction, which is a set of messages 1402 exchanged among objects 1404 within a collaboration to effect a desired operation or result. In a sequence diagram 1400, the vertical dimension represents time and the horizontal dimension represents different objects. A collaboration diagram 1500, depicted in Fig. 15, is also an interaction with messages 1502 exchanged among objects 1504, but it is also a collaboration, which is a set of objects 1504 related in

a particular context. Contrary to sequence diagrams 1400 (Fig. 14), which emphasize the time ordering of messages along the vertical axis, collaboration diagrams 1500 (Fig. 15) emphasize the structural organization of objects.

A statechart diagram 1600 is depicted in Fig. 16. The statechart diagram 1600 includes the sequences of states 1602 that an object or interaction goes through during its life in response to stimuli, together with its responses and actions. It uses a graphic notation that shows states of an object, the events that cause a transition from one state to another, and the actions that result from the transition.

The functional view can be represented by activity diagrams 1700 and more traditional descriptive narratives such as pseudocode and minispecifications. An activity diagram 1700 is depicted in Fig. 17, and is a special case of a state diagram where most, if not all, of the states are action states 1702 and where most, if not all, of the transitions are triggered by completion of the actions in the source states. Activity diagrams 1700 are used in situations where all or most of the events represent the completion of internally generated actions.

There is also a fourth view mingled with the static view called the architectural view. This view is modeled using package, component and deployment diagrams. Package diagrams show packages of classes and the dependencies among them. Component diagrams 1800, depicted in Fig. 18, are graphical representations of a system or its component parts. Component diagrams 1800 show the dependencies among software components, including source code components, binary code components and executable components. As depicted in Fig. 19, Deployment diagrams 1900 are used to show the distribution strategy for a distributed object system. Deployment diagrams 1900 show the configuration of run-time processing elements and the software components, processes and objects that live on them.

Although discussed in terms of class diagrams, one skilled in the art will recognize that the software development tool of the present invention may support these and other graphical views.

The improved software development tool of the present invention is used to display versions of source code. Each version is an instance in an edit history. The software development tool determines a language of the source code, stores indications of the edits to the source code, converts the source code with the indications of the edits from the

language into a language-neutral representation, uses the language-neutral representation to display the source code in the language with the indications of the edits, and uses the language-neutral representation to display the corresponding graphical representation of the source code with the indications of the edits. The rate at which the source code with the indications of the edits is displayed is adjustable. Moreover, the source code with the indications of the edits may be displayed sequentially or in reverse order.

Fig. 20 depicts a flow diagram of the steps performed by the software development tool to store the edit history of source code. After the source code has been modified (step 2000), the software development tool saves the edits to the source code into the secondary storage (step 2002). After all edits have been made (step 2004), the software development tool saves the source code into the secondary storage (step 2006).

The steps performed by the software development tool to sequentially display the source code with the edit history is depicted in Figs. 21A-C. The software development tool retrieves the source code (step 2100), identifies the programming language of the source code (step 2102), and obtains the template for the current programming language (step 2104). The software development tool then parses the source code (step 2106) and creates the data structure (step 2108). The software development tool retrieves the edit history, i.e., all of the edits which were stored for the source code (step 2110), and the user chooses a rate of displaying the source code (step 2112), thus setting the time period (step 2114). The user chooses whether to display the source code in the forward mode or in the reverse mode (step 2116).

In the forward mode shown in Fig. 21B, the software development tool removes all edits to the source code (step 2118) and updates the model (step 2120). The software development tool pauses for the time period determined by the rate at which the source code is displayed (step 2122), and for each edit (step 2124), the software development tool applies the edit to the source code (step 2126) before updating the model (step 2128). If the user chooses to adjust the rate of the display (step 2130), the time period is adjusted accordingly (step 2132). The process then continues with the next edit (step 2134).

In the reverse mode depicted in Fig. 21C, the software development tool pauses for the time period (step 2136), and for each edit (step 2138), the software development tool removes the edit from the source code (step 2140) before updating the model (step 2142). If the user chooses to adjust the rate of the display (step 2144), the software development

tool adjusts the time period accordingly (step 2146). The process then continues with the next edit (step 2148).

While various embodiments of the application have been described, it will be apparent to those of ordinary skill in the art that many more embodiments and implementations are possible that are within the scope of this invention. Accordingly, the invention is not to be restricted except in light of the attached claims and their equivalents.

Express Mail Label No. EV 339 908 705 US

CLAIMS

What is claimed is:

1. A method in a data processing system for displaying versions of source code, each version reflecting an instance in an edit history, the method comprising the steps of:

determining a language of the source code;

storing indications of the edits to the source code;

converting the source code with the indications of the edits from the language into a language-neutral representation;

using the language-neutral representation to display the source code in the language with the indications of the edits; and

using the language-neutral representation to display a corresponding graphical representation of the source code with the indications of the edits.

2. The method of claim 1, wherein the source code and the corresponding graphical representation of the source code are displayed sequentially.

3. The method of claim 1, wherein a rate at which the source code with the indications of the edits is displayed is adjustable.

4. The method of claim 1, wherein the source code with the indications of the edits is displayed in reverse order.

5. The method of claim 1, wherein the graphical representation is one of the group consisting of a class diagram, a use case diagram, a sequence diagram, a collaboration diagram, a state transition diagram, an activity diagram, a package diagram, a component diagram and a deployment diagram.

6. A method in a data processing system for displaying versions of source code, each version reflecting an instance in an edit history, the method comprising the steps of:

storing indications of the edits to the source code; and

displaying the versions of the source code with the indications of the edits.

7. The method of claim 6, wherein the versions of the source code are displayed sequentially.

8. The method of claim 6, wherein a rate at which the source code with the indications of the edits is displayed is adjustable.

9. The method of claim 6, wherein the source code with the indications of the edits is displayed in reverse order.

10. The method of claim 6, wherein the versions of the source code are displayed with a corresponding graphical representation for each version.

11. The method of claim 10, wherein the step of displaying the versions of source code comprises the steps of:

determining a language of the source code;

converting the source code with the indications of the edits from the language into a language-neutral representation;

using the language-neutral representation to display the source code in the language with the indications of the edits; and

using the language-neutral representation to display the corresponding graphical representation of the source code with the indications of the edits.

12. The method of claim 10, wherein the graphical representation is one of the group consisting of a class diagram, a use case diagram, a sequence diagram, a collaboration diagram, a state transition diagram, an activity diagram, a package diagram, a component diagram and a deployment diagram.

13. A method in a data processing system for displaying versions of source code, the method comprising the steps of:

storing an edit to the source code;

displaying the source code and a graphical representation of the source code; and

displaying the source code with the edit and a graphical representation of the source code with the edit.

14. The method of claim 13, wherein the step of displaying the source code comprises the steps of:

determining a language of the source code;

converting the source code from the language into a language-neutral representation; and

using the language-neutral representation to display the graphical representation of the source code.

15. The method of claim 13, wherein the step of displaying the source code with the edit comprises the steps of:

converting the source code with the edit from the language into a language-neutral representation; and

using the language-neutral representation of the source code with the edit to display the graphical representation of the source code with the edit.

16. The method of claim 13, wherein the source code is displayed after the source code with the edit is displayed.

17. A computer-readable medium containing instructions for controlling a data processing system to perform a method, the data processing system having versions of source code, each version reflecting an instance in an edit history, the method comprising the steps of:

determining a language of the source code;

storing indications of the edits to the source code;

converting the source code with the indications of the edits from the language into a language-neutral representation;

using the language-neutral representation to display the source code in the language with the indications of the edits; and

using the language-neutral representation to display a corresponding graphical representation of the source code with the indications of the edits.

18. The computer-readable medium of claim 17, wherein the source code and the corresponding graphical representation of the source code are displayed sequentially.

19. The computer-readable medium of claim 17, wherein a rate at which the source code with the indications of the edits is displayed is adjustable.

20. The computer-readable medium of claim 17, wherein the source code with the indications of the edits is displayed in reverse order.

21. The computer-readable medium of claim 17, wherein the graphical representation is one of the group consisting of a class diagram, a use case diagram, a sequence diagram, a collaboration diagram, a state transition diagram, an activity diagram, a package diagram, a component diagram and a deployment diagram.

22. A computer-readable medium containing instructions for controlling a data processing system to perform a method, the data processing system having versions of source code, each version reflecting an instance in an edit history, the method comprising the steps of:

storing indications of edits to the source code; and

displaying the versions of the source code with the indications of the edits.

23. The computer-readable medium of claim 22, wherein the versions of the source code are displayed sequentially.

24. The computer-readable medium of claim 22, wherein a rate at which the source code with the indications of the edits is displayed is adjustable.

25. The computer-readable medium of claim 22, wherein the source code with the indications of the edits is displayed in reverse order.

26. The computer-readable medium of claim 22, wherein the versions of the source code are displayed with a corresponding graphical representation for each version.

27. The computer-readable medium of claim 26, wherein the step of displaying the versions of source code comprises the steps of:

determining a language of the source code;

converting the source code with the indications of the edits from the language into a language-neutral representation;

using the language-neutral representation to display the source code in the language with the indications of the edits; and

using the language-neutral representation to display the corresponding graphical representation of the source code with the indications of the edits.

28. The computer-readable medium of claim 26, wherein the graphical representation is one of the group consisting of a class diagram, a use case diagram, a sequence diagram, a collaboration diagram, a state transition diagram, an activity diagram, a package diagram, a component diagram and a deployment diagram.

29. A computer-readable medium containing instructions for controlling a data processing system to perform a method, the data processing system having source code, the method comprising the steps of:

storing an edit to the source code;

displaying the source code and a graphical representation of the source code; and

displaying the source code with the edit and a graphical representation of the source code with the edit.

30. The computer-readable medium of claim 29, wherein the step of displaying the source code comprises the steps of:

determining a language of the source code;

converting the source code from the language into a language-neutral representation; and

using the language-neutral representation to display the graphical representation of the source code.

31. The computer-readable medium of claim 29, wherein the step of displaying the source code with the edit comprises the steps of:

converting the source code with the edit from the language into a language-neutral representation; and

using the language-neutral representation of the source code with the edit to display the graphical representation of the source code with the edit.

32. The computer-readable medium of claim 29, wherein the source code is displayed after the source code with the edit is displayed.

33. A data processing system comprising:

a secondary storage including source code;

a memory device including:

a program that stores indications of edits to the source code into the memory device, and that displays the source code with the indications of the edits and a corresponding graphical representation of the source code with the indications of the edits; and

a processor for running the program.

34. The data processing system of claim 33, wherein the source code with the indications of the edits are displayed sequentially.

35. The data processing system of claim 33, wherein a rate at which the source code with the indications of the edits is displayed is adjustable.

36. The data processing system of claim 33, wherein the source code with the indications of the edits is displayed in reverse order.

37. The data processing system of claim 33, wherein the program further determines the language of the source code, converts the source code with the indications of the edits from the language into a language-neutral representation, uses the language-neutral representation to display the source code with the indications of the edits in the language, and uses the language-neutral representation to display the corresponding graphical representation of the source code with the indications of the edits.

38. The data processing system of claim 37, wherein the memory device further comprises a transient meta model, wherein said transient meta model stores the language-neutral representation of the source code.

39. The data processing system of claim 33, wherein the graphical representation is one of the group consisting of a class diagram, a use case diagram, a sequence diagram, a collaboration diagram, a state transition diagram, an activity diagram, a package diagram, a component diagram and a deployment diagram.

40. A system for displaying versions of source code, each version reflecting an instance in an edit history, the system comprising:

means for storing indications of the edits to the source code; and

means for displaying the versions of the source code with the indications of the edits.

0604-0074 5960-5960

ABSTRACT OF THE DISCLOSURE

Methods and systems consistent with the present invention provide an improved software development tool which displays versions of source code. Each version reflects an instance in an edit history, i.e., reflects the changes made to the source code.

0604P-050US

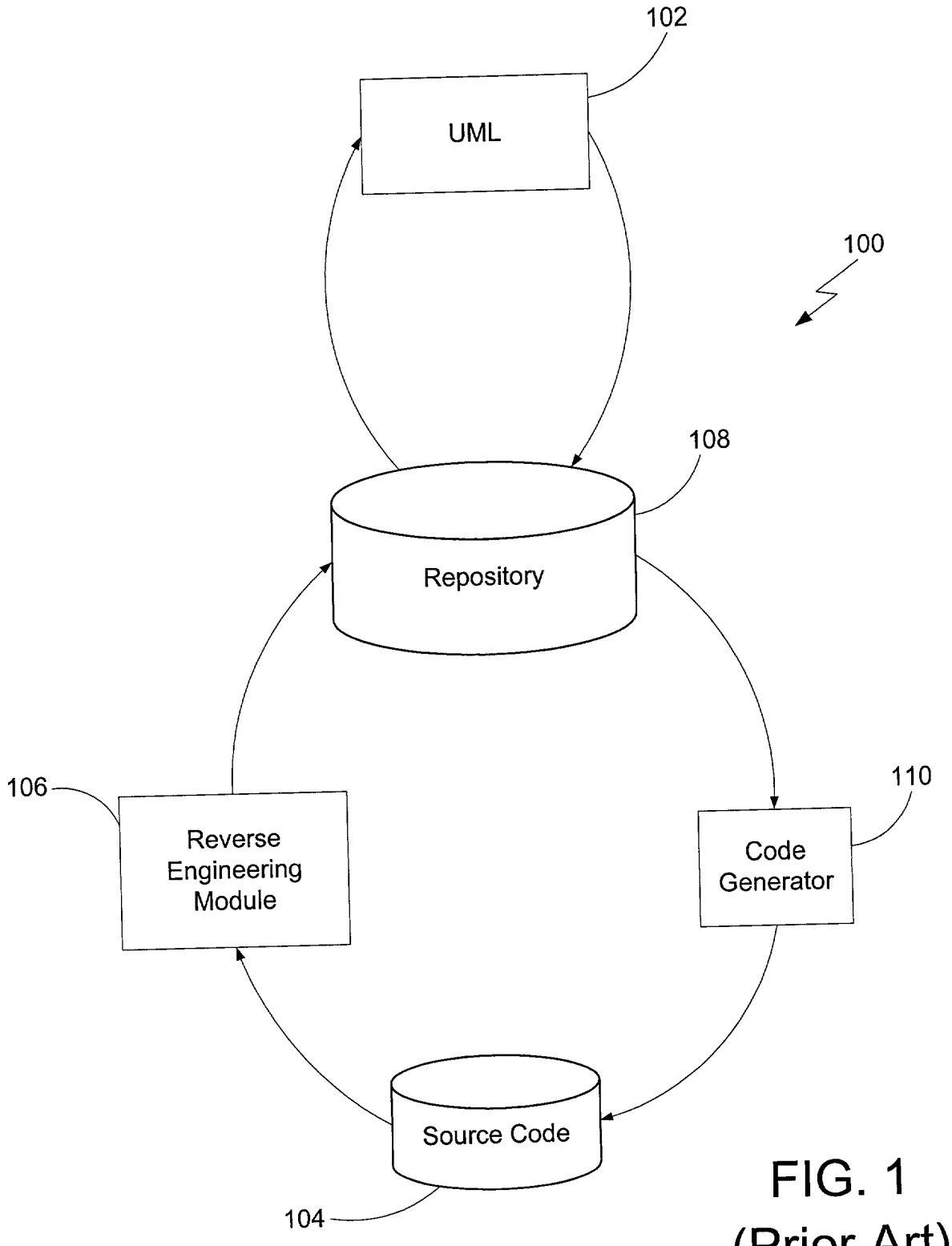

FIG. 1
(Prior Art)

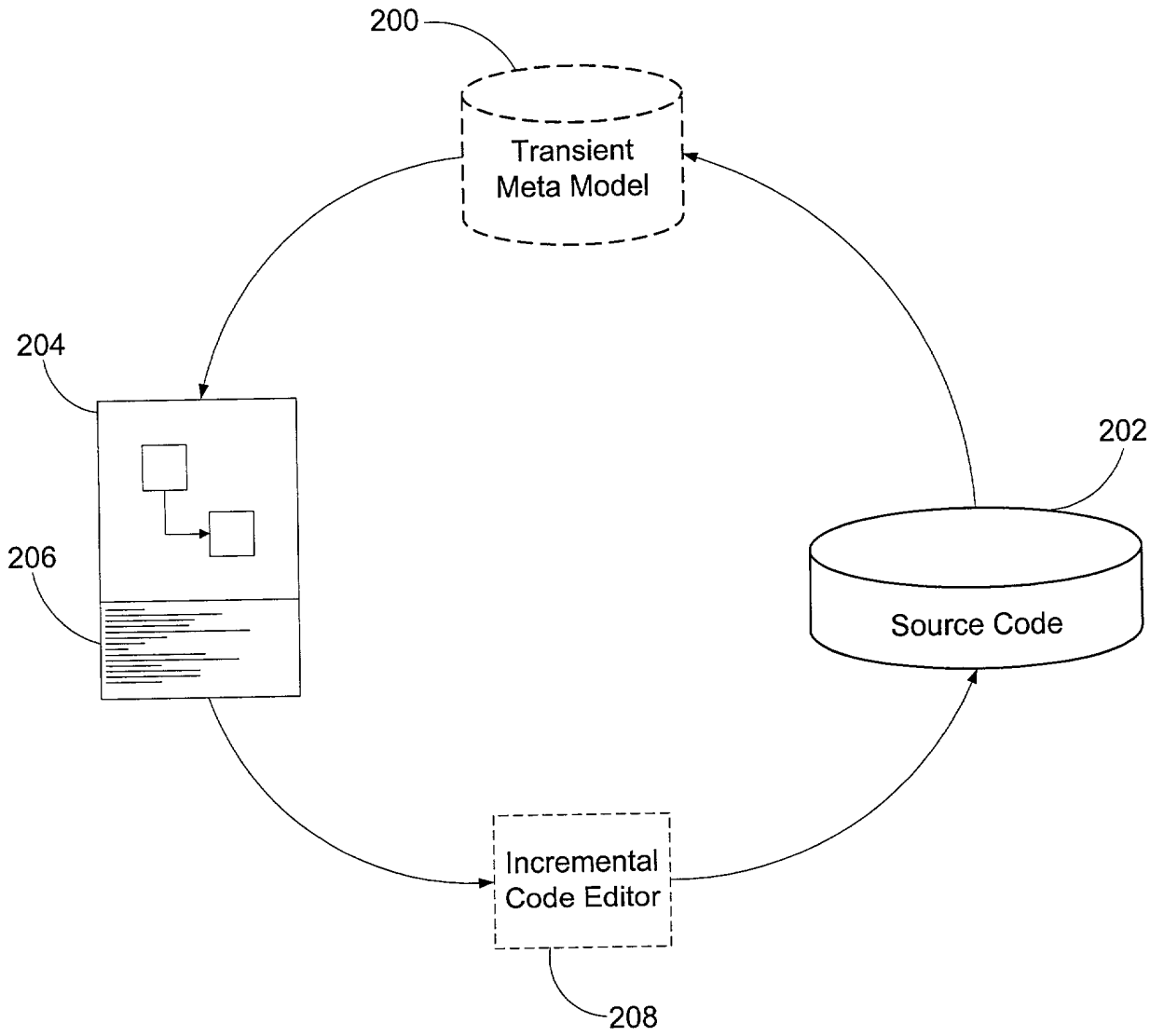
200
Transient
Meta Model
204
206
202
Source Code
Incremental
Code Editor
208

FIG. 2

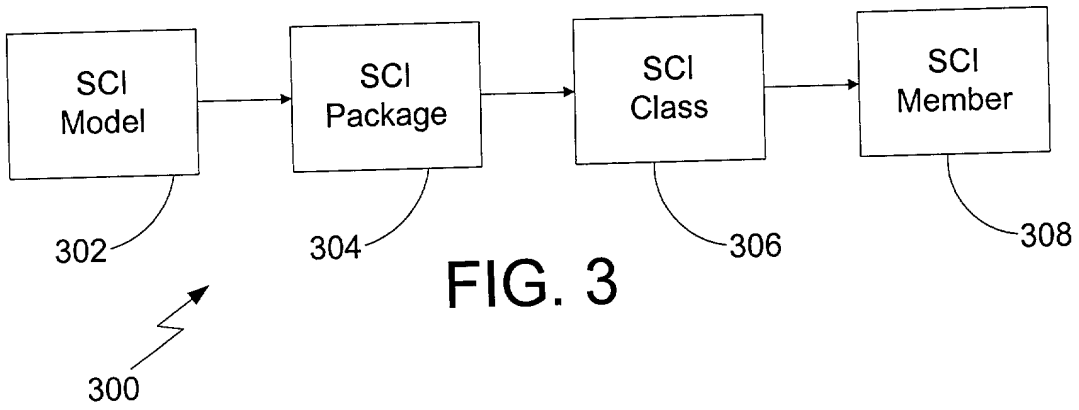

FIG. 3

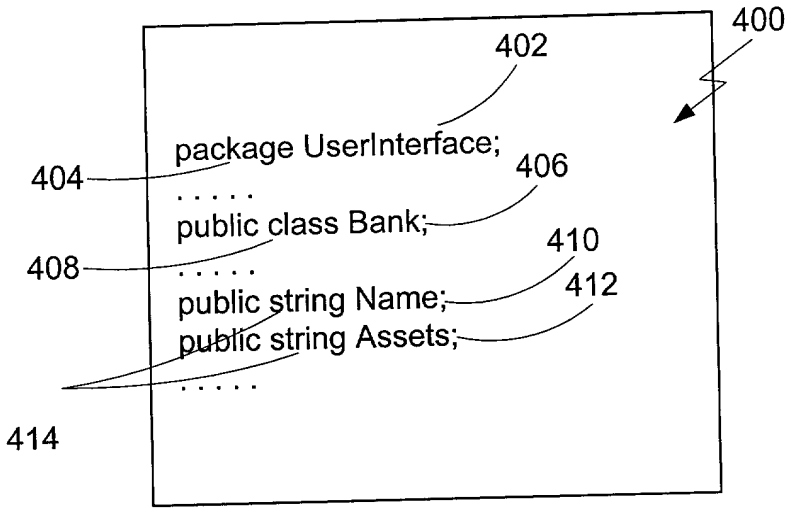

FIG. 4

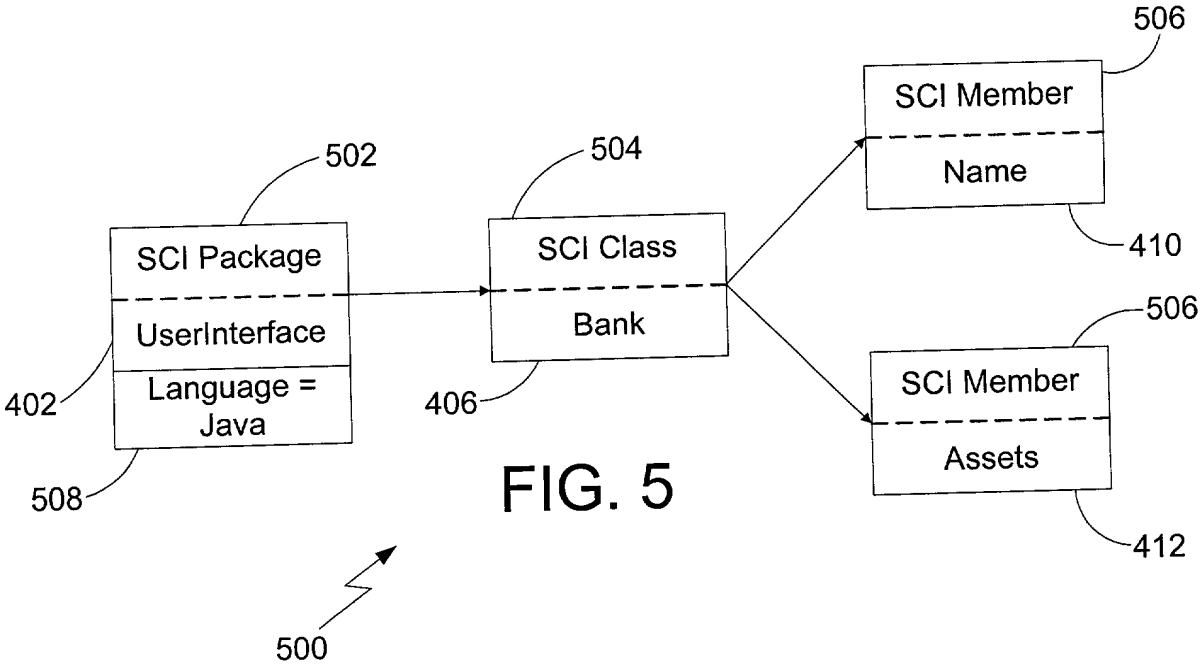

FIG. 5

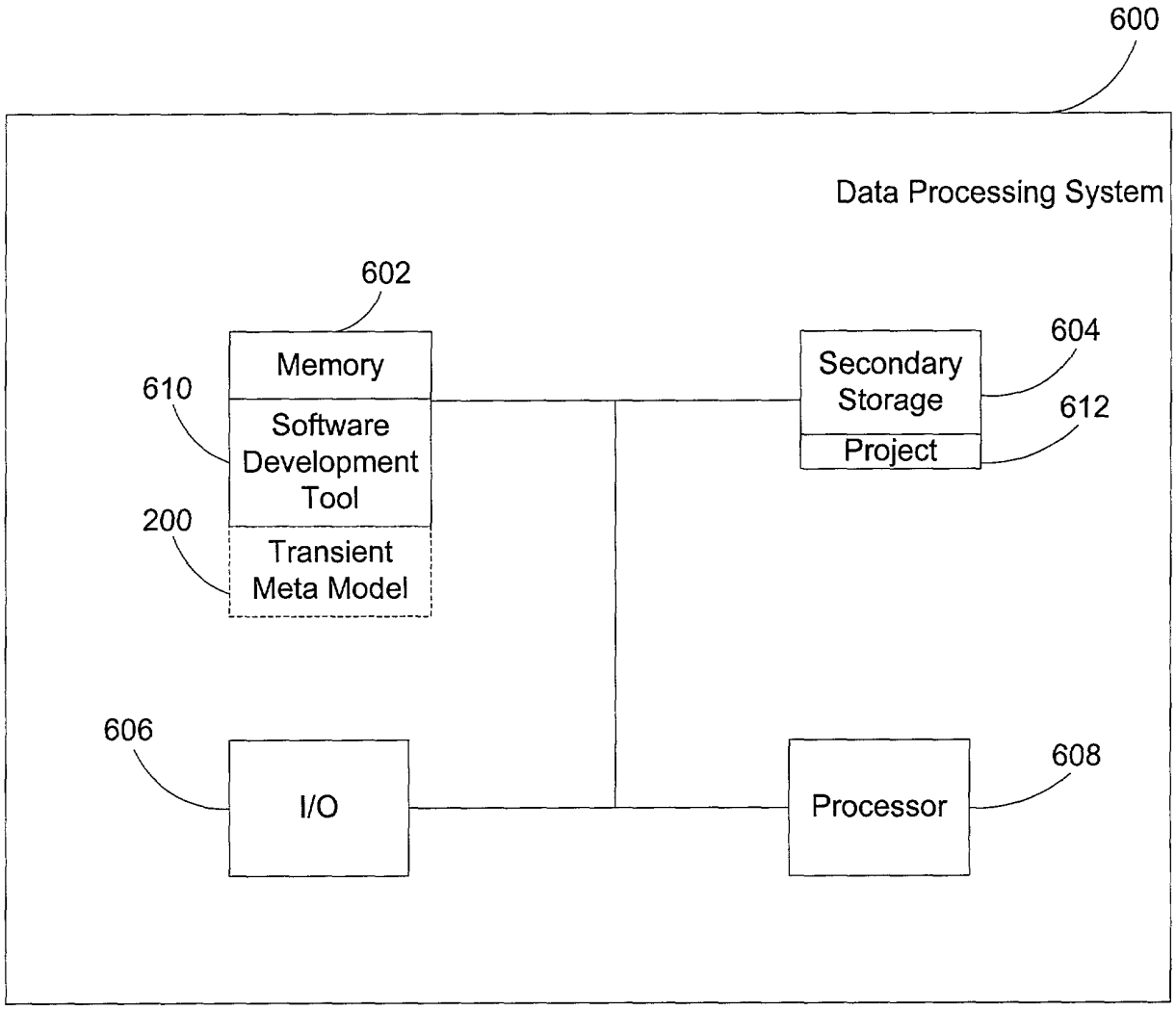
600
Data Processing System
602
Memory
610
Software Development Tool
200
Transient Meta Model
604
Secondary Storage
612
Project
606
I/O
608
Processor

FIG. 6

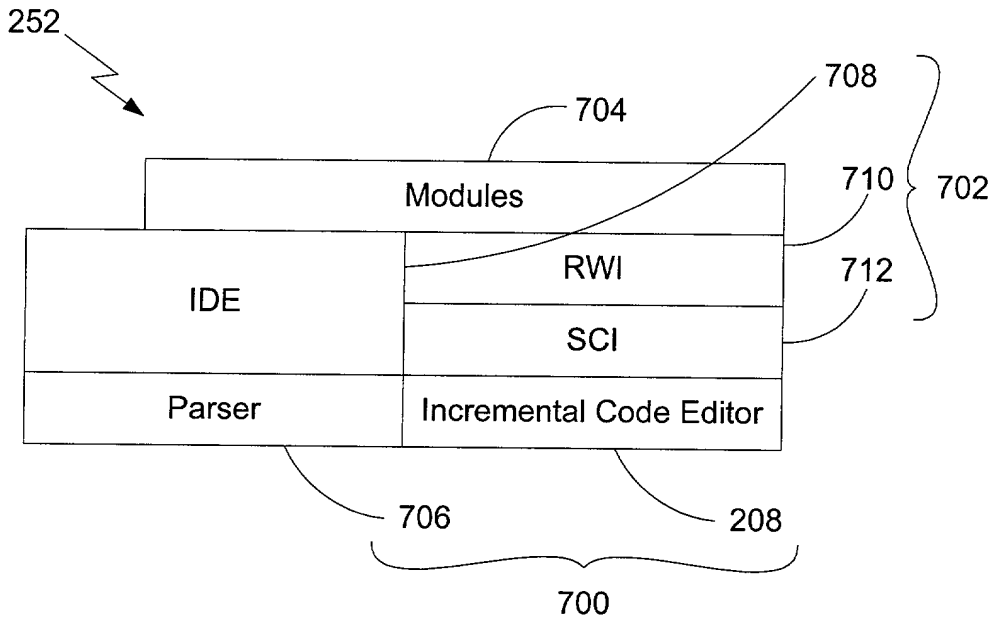
252
708
704
Modules
710
702
RWI
712
IDE
SCI
Parser
Incremental Code Editor
706
208
700

FIG. 7

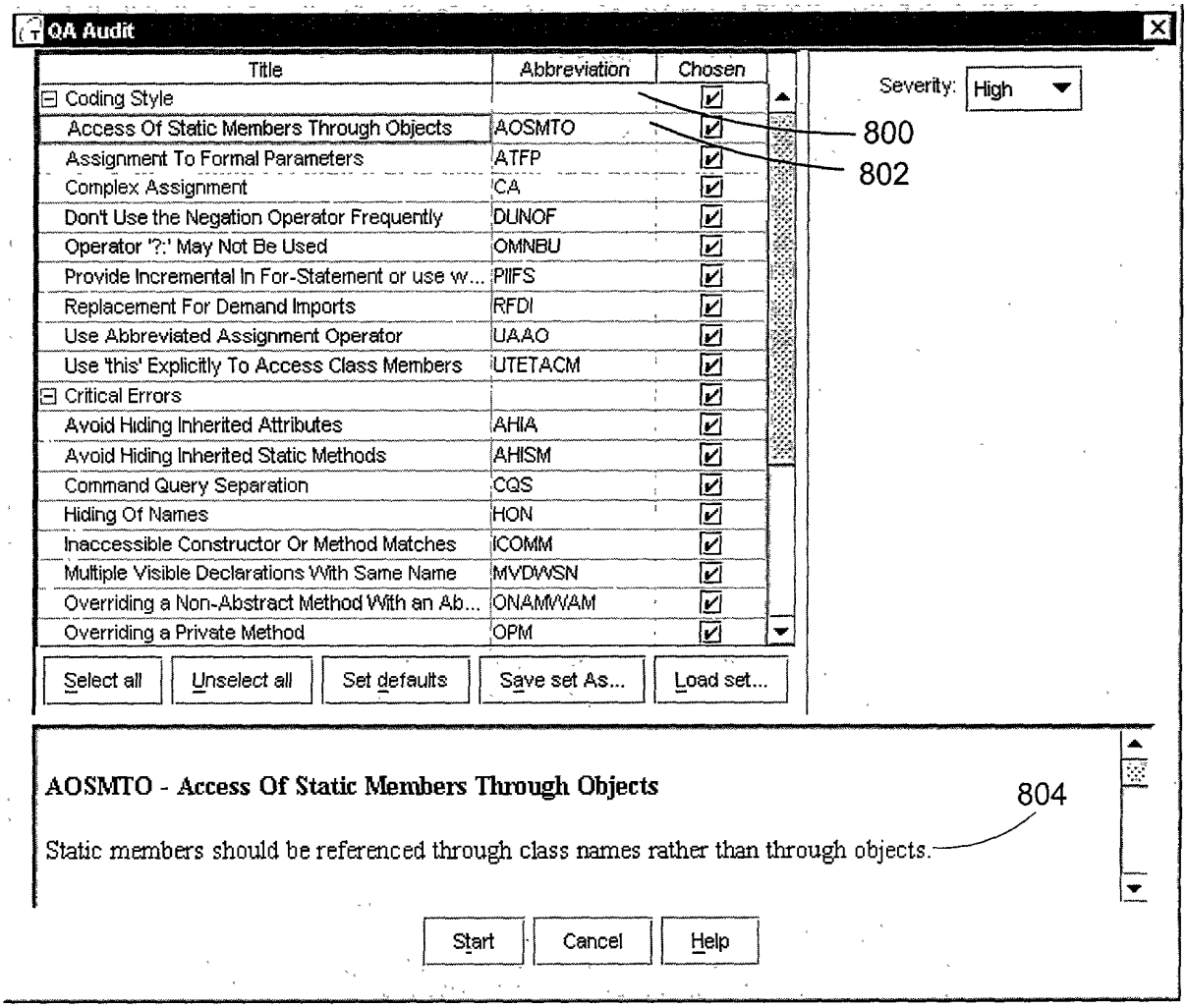
QA Audit
Title
Abbreviation
Chosen
Coding Style
Access Of Static Members Through Objects
AOSMTO
Assignment To Formal Parameters
ATFP
Complex Assignment
CA
Don't Use the Negation Operator Frequently
DUNOF
Operator '?:' May Not Be Used
OMNBU
Provide Incremental In For-Statement or use w...
PIIFS
Replacement For Demand Imports
RFDI
Use Abbreviated Assignment Operator
UAAO
Use 'this' Explicitly To Access Class Members
UTETACM
Critical Errors
Avoid Hiding Inherited Attributes
AHIA
Avoid Hiding Inherited Static Methods
AHISM
Command Query Separation
CQS
Hiding Of Names
HON
Inaccessible Constructor Or Method Matches
ICOMM
Multiple Visible Declarations With Same Name
MVDWSN
Overriding a Non-Abstract Method With an Ab...
ONAMWAM
Overriding a Private Method
OPM
Severity:
High
800
802
Select all
Unselect all
Set defaults
Save set As...
Load set...
AOSMTO - Access Of Static Members Through Objects
804
Static members should be referenced through class names rather than through objects.
Start
Cancel
Help

FIG. 8A

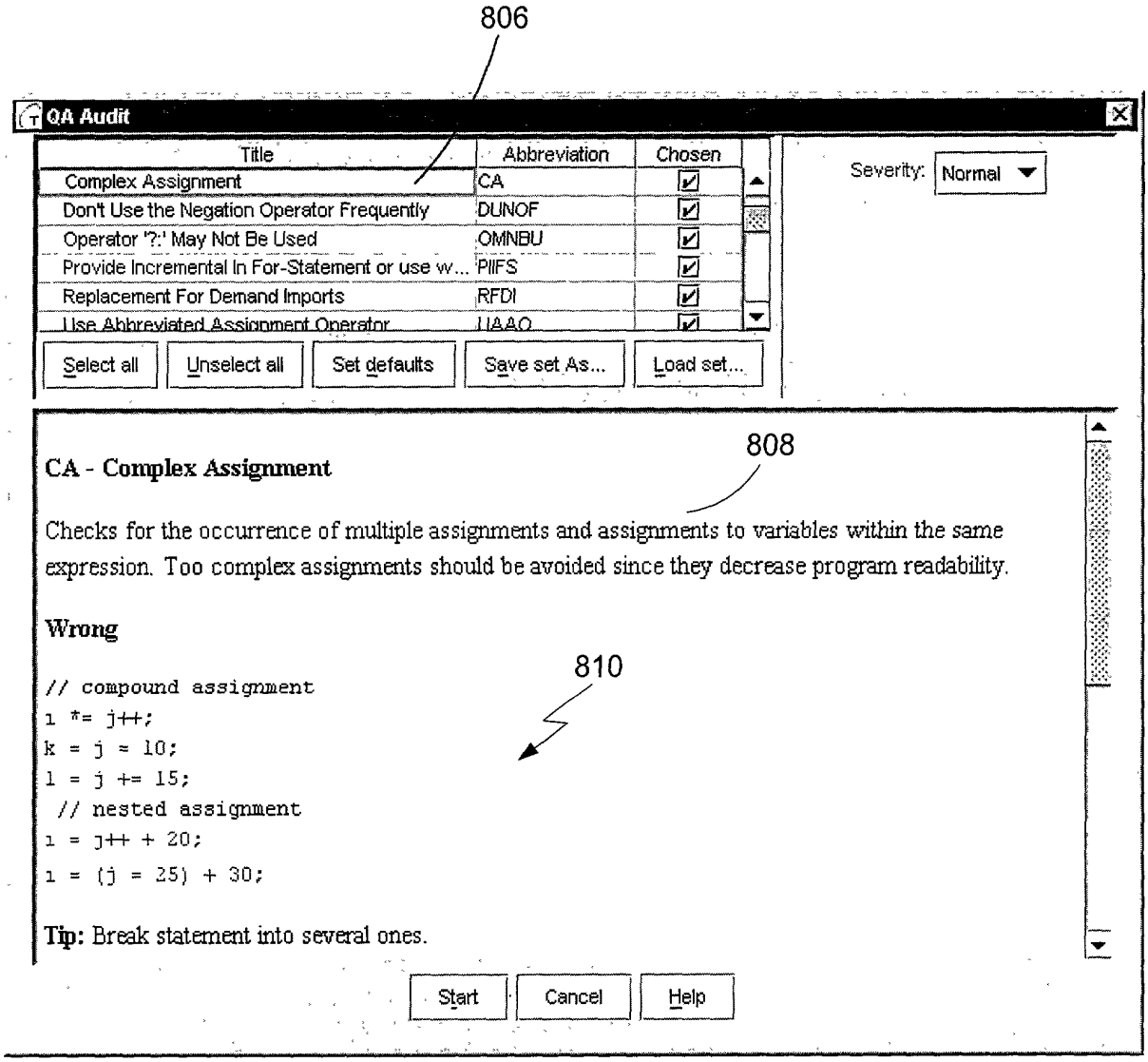
806
QA Audit
Title
Abbreviation
Chosen
Complex Assignment
CA
Don't Use the Negation Operator Frequently
DUNOF
Operator '?:' May Not Be Used
OMNBU
Provide Incremental In For-Statement or use w...
PIIFS
Replacement For Demand Imports
RFDI
Use Abbreviated Assignment Operator
UAAO
Severity:
Normal
Select all
Unselect all
Set defaults
Save set As...
Load set...
808
CA - Complex Assignment
Checks for the occurrence of multiple assignments and assignments to variables within the same expression. Too complex assignments should be avoided since they decrease program readability.
Wrong
810
// compound assignment
i *= j++;
k = j = 10;
l = j += 15;
// nested assignment
i = j++ + 20;
i = (j = 25) + 30;
Tip: Break statement into several ones.
Start
Cancel
Help

FIG. 8B

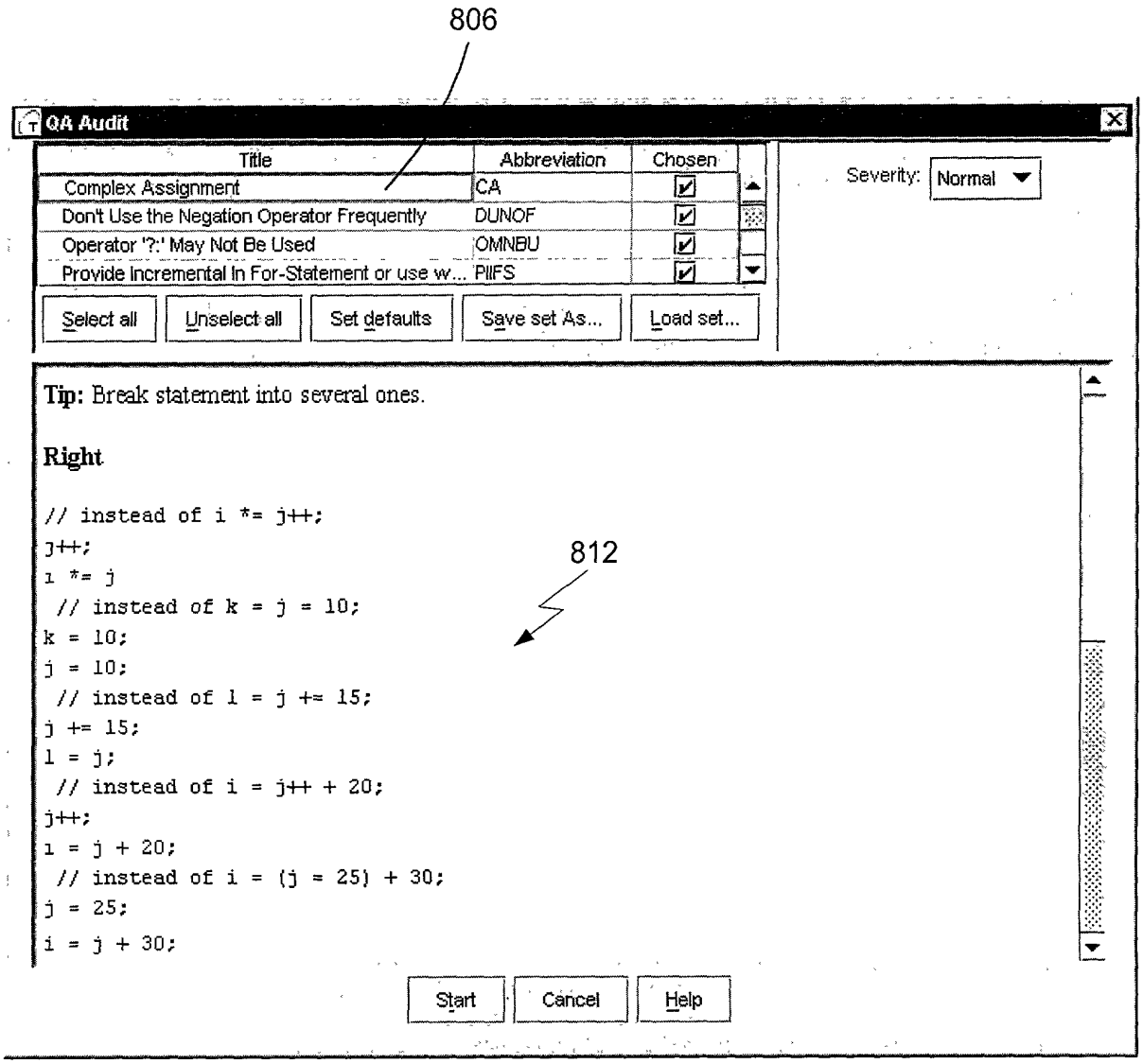
806
QA Audit
Title
Abbreviation
Chosen
Complex Assignment
CA
Don't Use the Negation Operator Frequently
DUNOF
Operator '?:' May Not Be Used
OMNBU
Provide Incremental In For-Statement or use w...
PIIFS
Severity:
Normal
Select all
Unselect all
Set defaults
Save set As...
Load set...
Tip: Break statement into several ones.
Right
// instead of i *= j++;
j++;
i *= j
// instead of k = j = 10;
k = 10;
j = 10;
// instead of l = j += 15;
j += 15;
l = j;
// instead of i = j++ + 20;
j++;
i = j + 20;
// instead of i = (j = 25) + 30;
j = 25;
i = j + 30;
812
Start
Cancel
Help

FIG. 8C

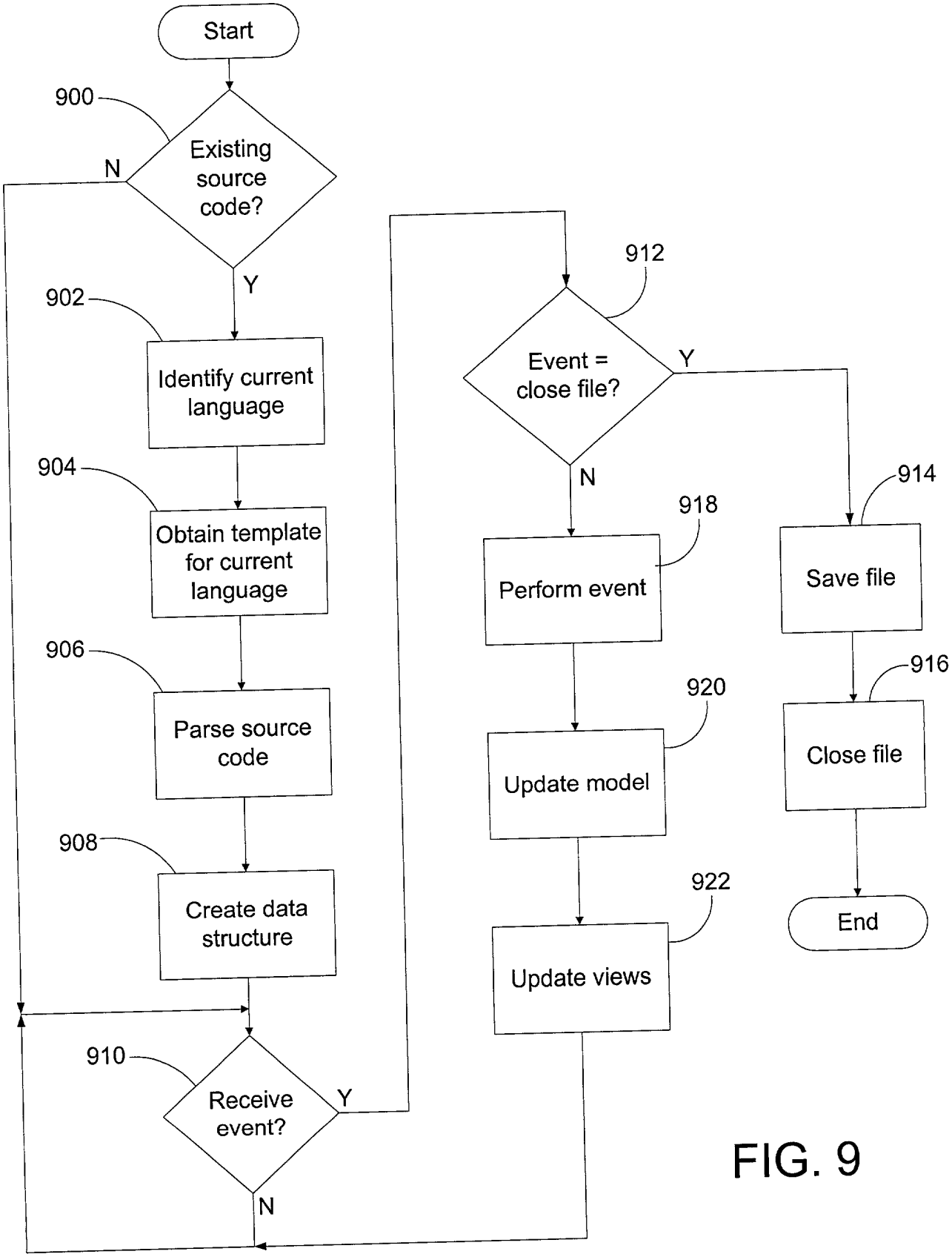
Start
900
Existing source code?
N
Y
902
Identify current language
904
Obtain template for current language
906
Parse source code
908
Create data structure
910
Receive event?
Y
N
912
Event = close file?
Y
N
914
Save file
916
Close file
End
918
Perform event
920
Update model
922
Update views

FIG. 9

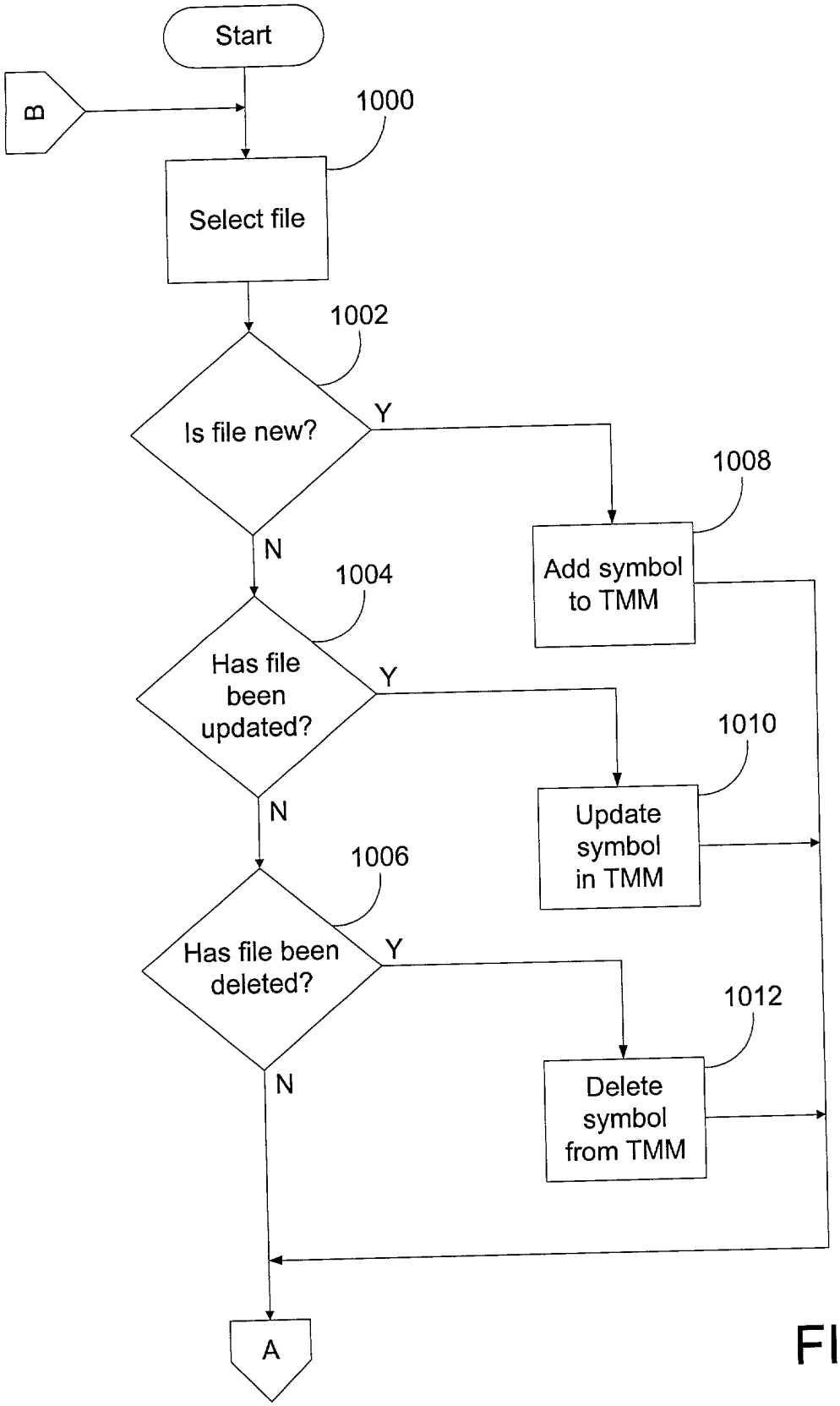
Start
B
1000
Select file
1002
Is file new?
Y
N
1008
Add symbol to TMM
1004
Has file been updated?
Y
N
1010
Update symbol in TMM
1006
Has file been deleted?
Y
N
1012
Delete symbol from TMM
A

FIG. 10A

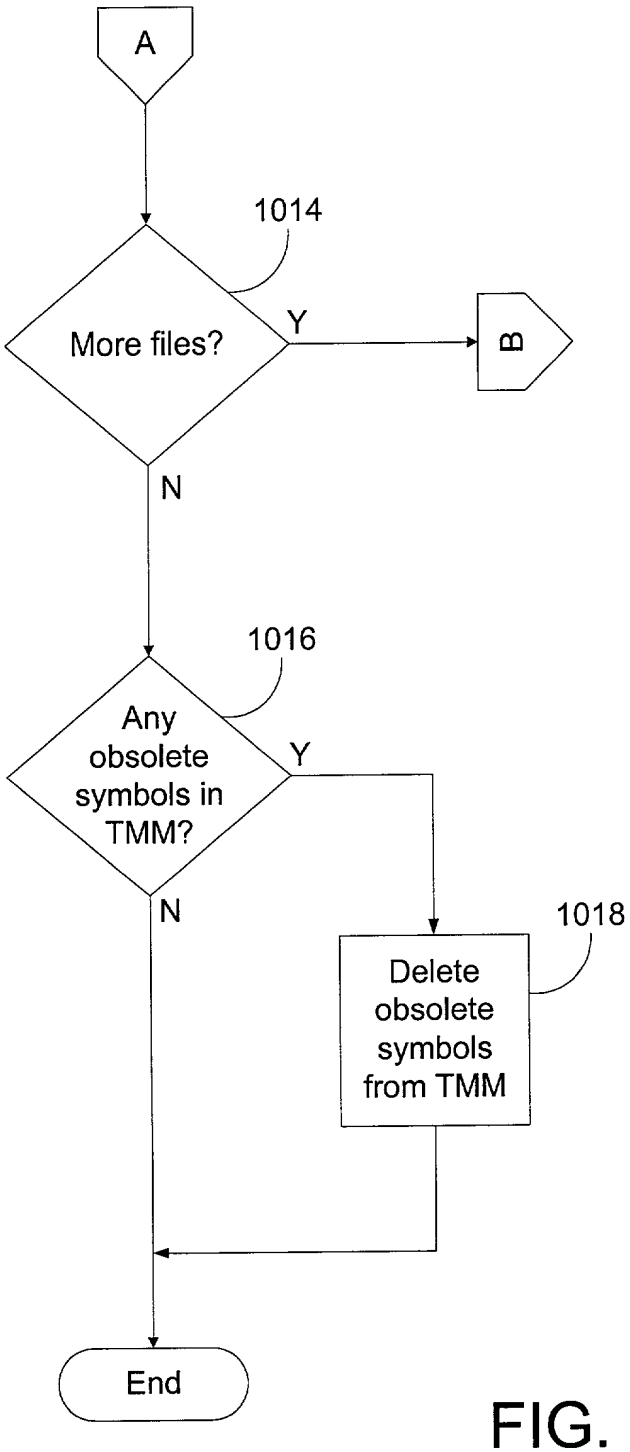
A
1014
More files?
Y
B
N
1016
Any obsolete symbols in TMM?
Y
N
1018
Delete obsolete symbols from TMM
End

FIG. 10B

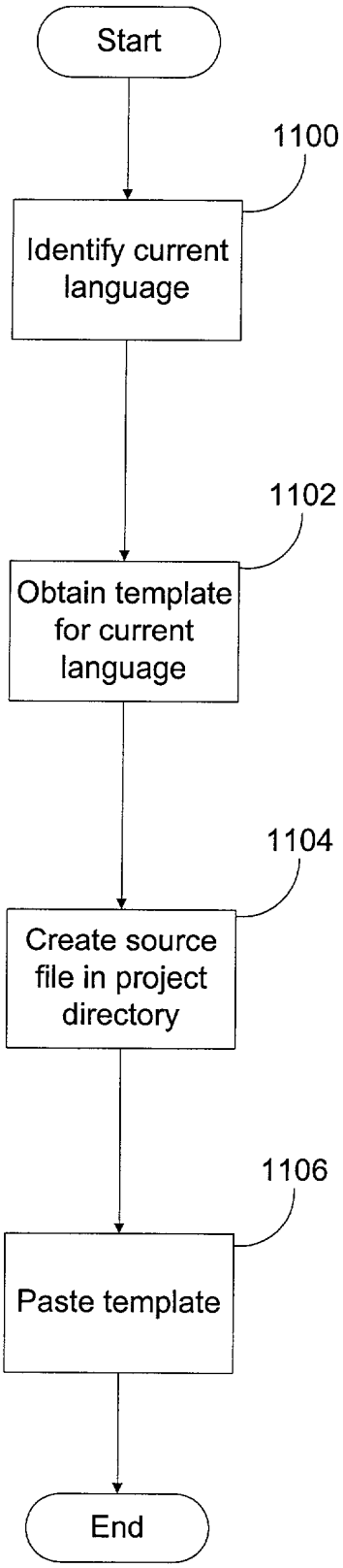
Start
1100
Identify current language
1102
Obtain template for current language
1104
Create source file in project directory
1106
Paste template
End

FIG. 11

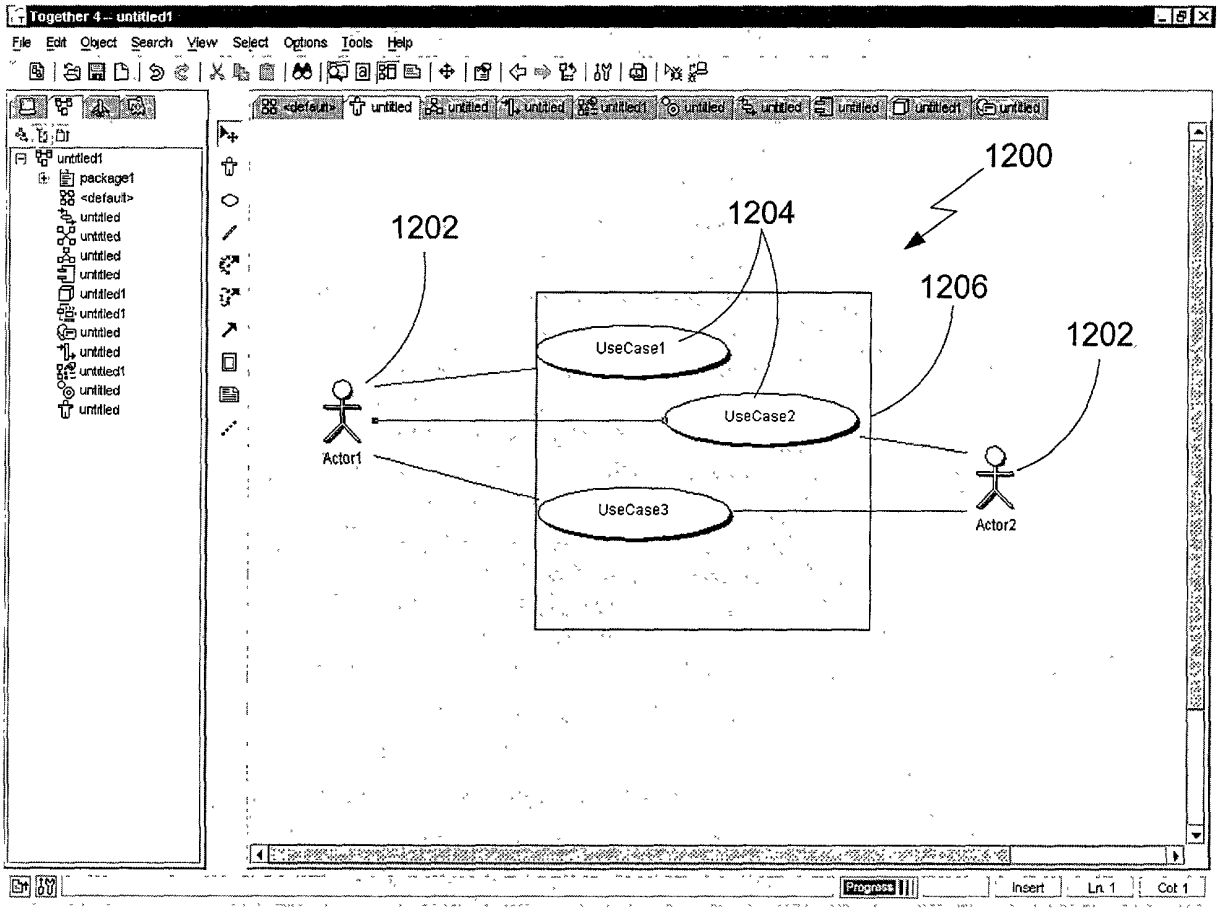
Together 4 -- untitled1
File Edit Object Search View Select Options Tools Help
untitled1
package1
<default>
untitled
untitled
untitled
untitled
untitled1
untitled1
untitled
untitled
untitled1
untitled
untitled
1200
1202
1204
1206
1202
UseCase1
UseCase2
UseCase3
Actor1
Actor2
Insert
Ln 1
Col 1

FIG. 12

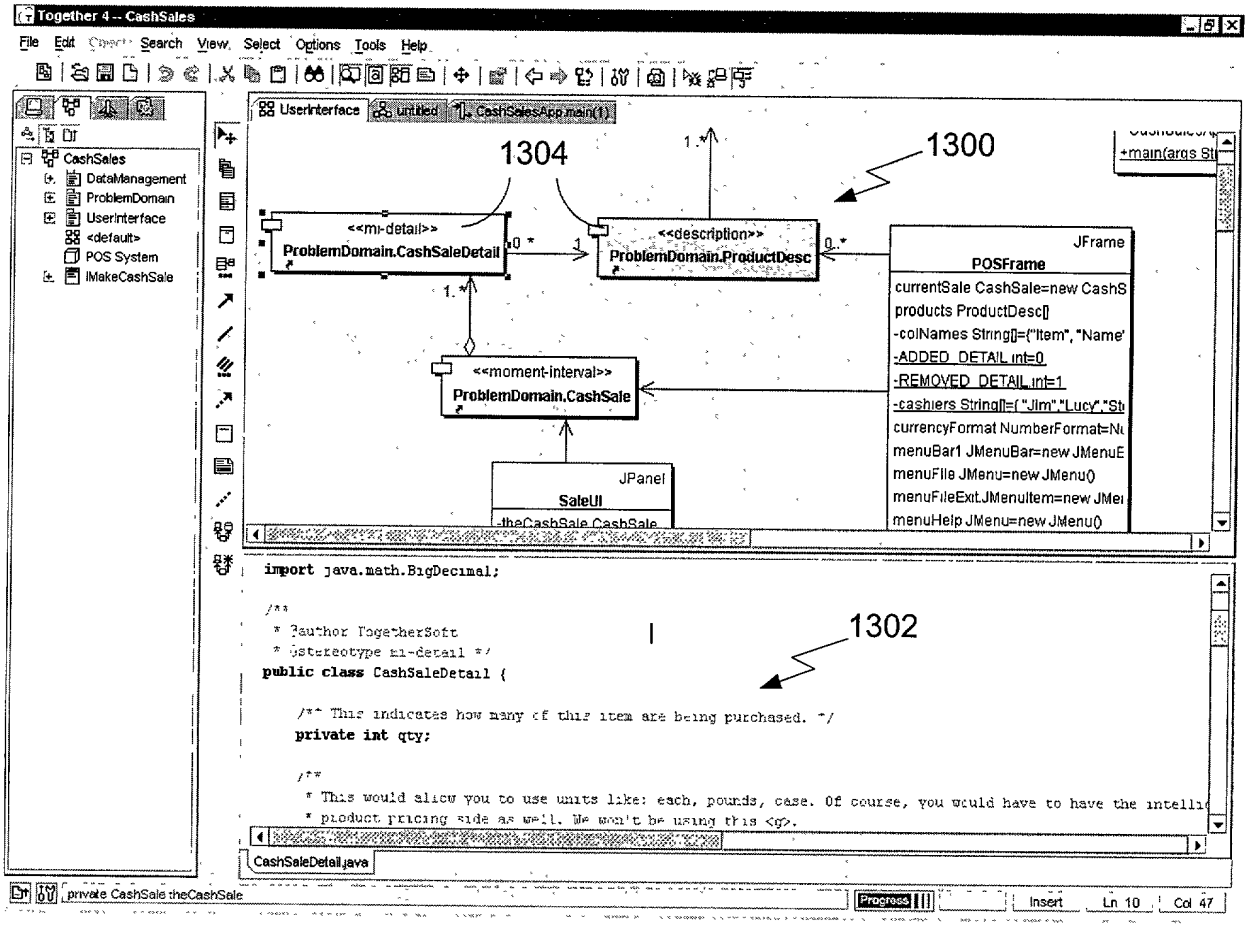
Together 4 -- CashSales
File Edit Search View Select Options Tools Help
CashSales
DataManagement
ProblemDomain
UserInterface
<default>
POS System
IMakeCashSale
UserInterface
untitled
CashSalesApp.main(1)
1304
1300
<<mi-detail>>
ProblemDomain.CashSaleDetail
<<description>>
ProblemDomain.ProductDesc
<<moment-interval>>
ProblemDomain.CashSale
JFrame
POSFrame
currentSale CashSale=new CashS
products ProductDesc[]
-colNames String[]={"Item","Name"
-ADDED_DETAIL int=0
-REMOVED_DETAIL int=1
currencyFormat NumberFormat=N
menuBar1 JMenuBar=new JMenuE
menuFile JMenu=new JMenu()
menuHelp JMenu=new JMenu()
JPanel
SaleUI
1302
import java.math.BigDecimal;
* @author TogetherSoft
* @stereotype mi-detail */
public class CashSaleDetail {
/** This indicates how many of this item are being purchased. */
private int qty;
CashSaleDetail.java
private CashSale theCashSale
Progress
Insert
Ln 10
Col 47

FIG. 13

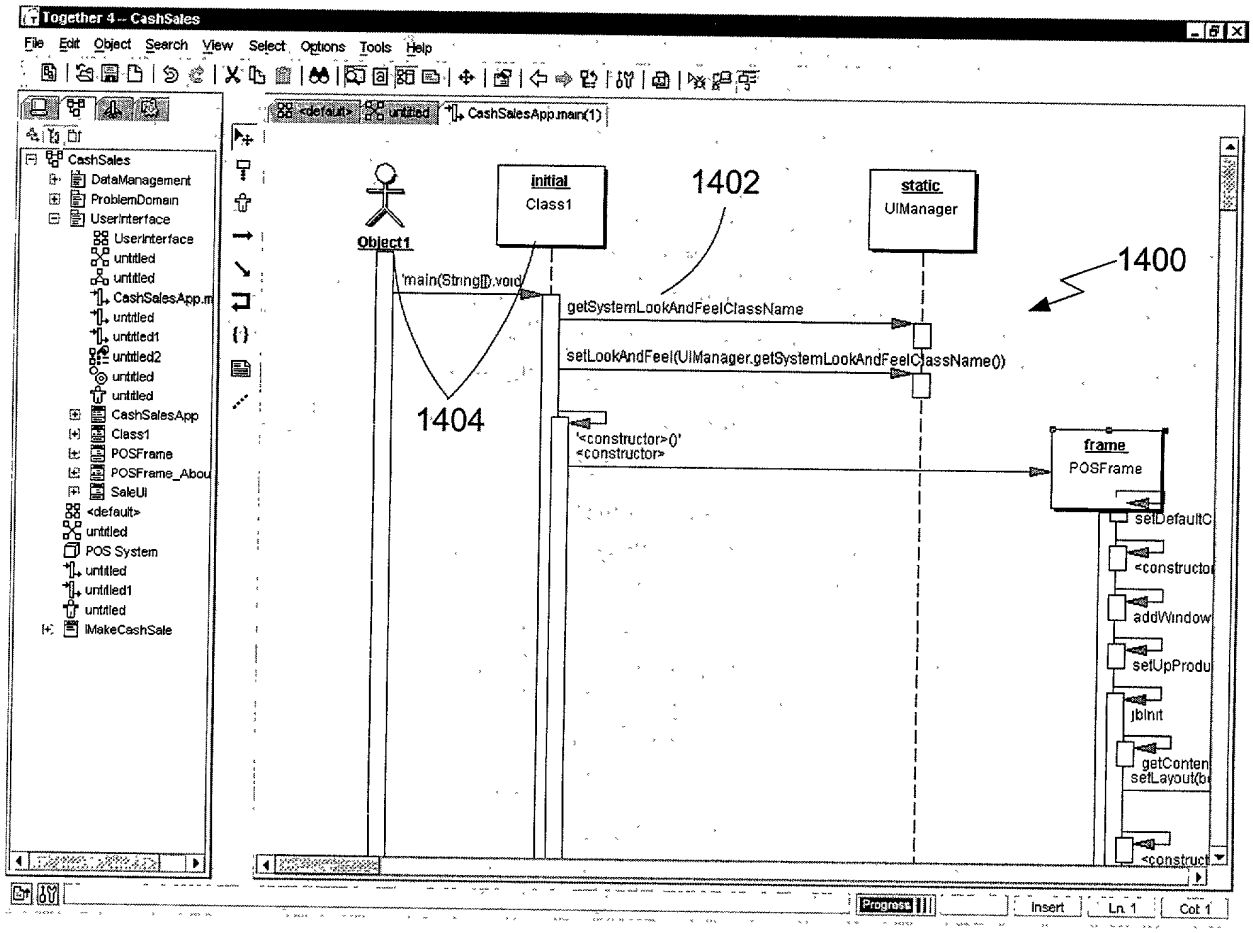
Together 4 -- CashSales
File Edit Object Search View Select Options Tools Help
CashSales
DataManagement
ProblemDomain
UserInterface
UserInterface
untitled
untitled
CashSalesApp.m
untitled
untitled1
untitled2
untitled
untitled
CashSalesApp
Class1
POSFrame
POSFrame_Abou
SaleUI
<default>
untitled
POS System
untitled
untitled1
untitled
IMakeCashSale
<default>
untitled
CashSalesApp.main(1)
initial
Class1
1402
static
UIManager
1400
Object1
'main(String[]).void'
getSystemLookAndFeelClassName
setLookAndFeel(UIManager.getSystemLookAndFeelClassName())
1404
'<constructor>()'
<constructor>
frame
POSFrame
setDefaultC
<constructo
addWindow
setUpProdu
jbInit
getConten
setLayout(b
<construct
Progress
Insert
Ln 1
Col 1

FIG. 14

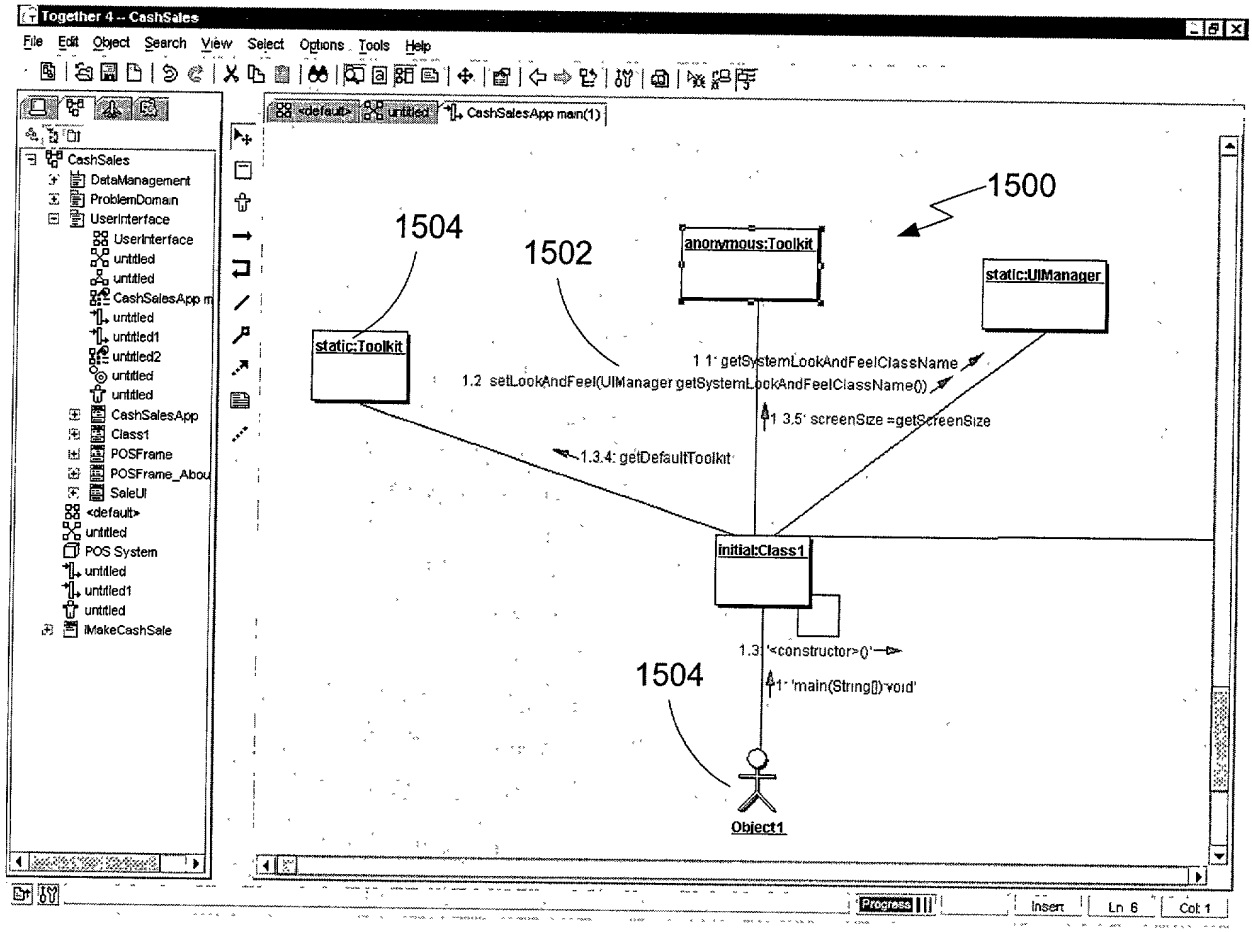
Together 4 -- CashSales
File Edit Object Search View Select Options Tools Help
CashSales
DataManagement
ProblemDomain
UserInterface
UserInterface
untitled
untitled
CashSalesApp m
untitled
untitled1
untitled2
untitled
untitled
CashSalesApp
Class1
POSFrame
POSFrame_Abou
SaleUI
<default>
untitled
POS System
untitled
untitled1
untitled
IMakeCashSale
<default>
untitled
CashSalesApp main(1)
1500
1504
1502
anonymous:Toolkit
static:UIManager
static:Toolkit
1 1: getSystemLookAndFeelClassName
1.2 setLookAndFeel(UIManager.getSystemLookAndFeelClassName())
1 3.5: screenSize =getScreenSize
1.3.4: getDefaultToolkit
initial:Class1
1.3: '<constructor>()'
1504
1: 'main(String[])'void'
Object1
Progress
Insert
Ln 6
Col 1

FIG. 15

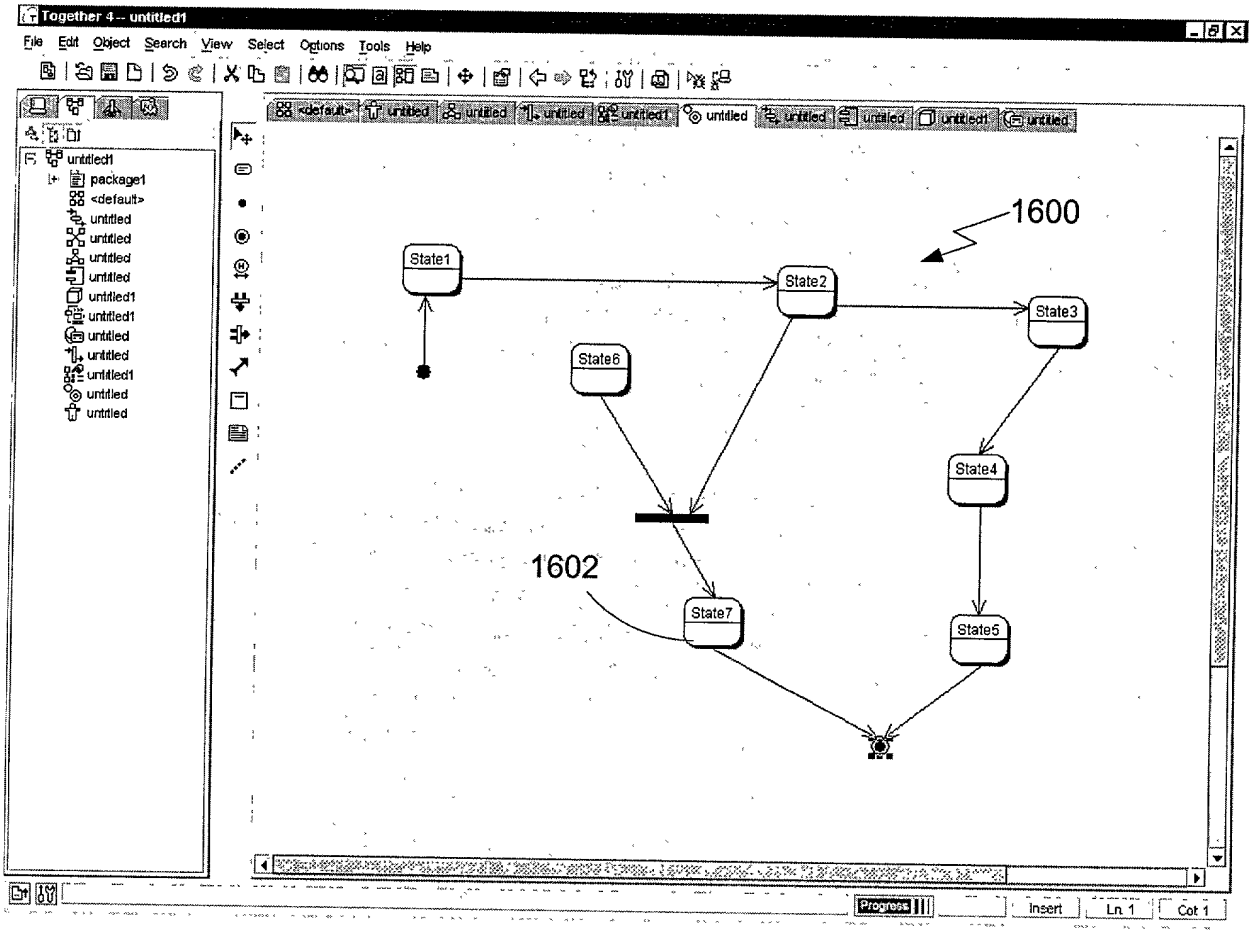
Together 4-- untitled1
File Edit Object Search View Select Options Tools Help
untitled1
package1
<default>
1600
State1
State2
State3
State4
State5
State6
State7
1602
Insert
Ln 1
Col 1

FIG. 16

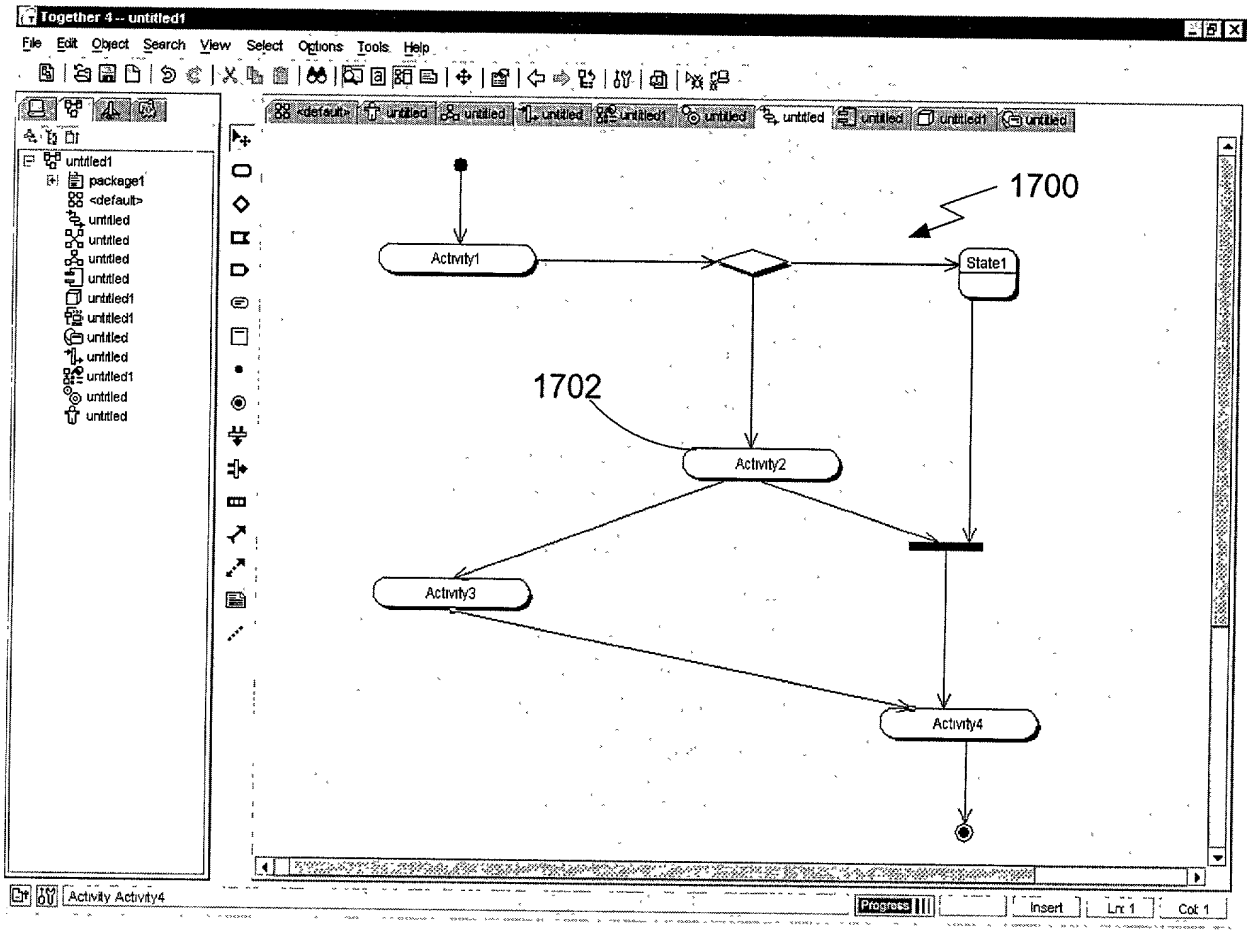
Together 4 -- untitled1
File Edit Object Search View Select Options Tools Help
untitled1
package1
<default>
1700
Activity1
State1
1702
Activity2
Activity3
Activity4
Activity Activity4
Progress
Insert
Ln: 1
Col: 1

FIG. 17

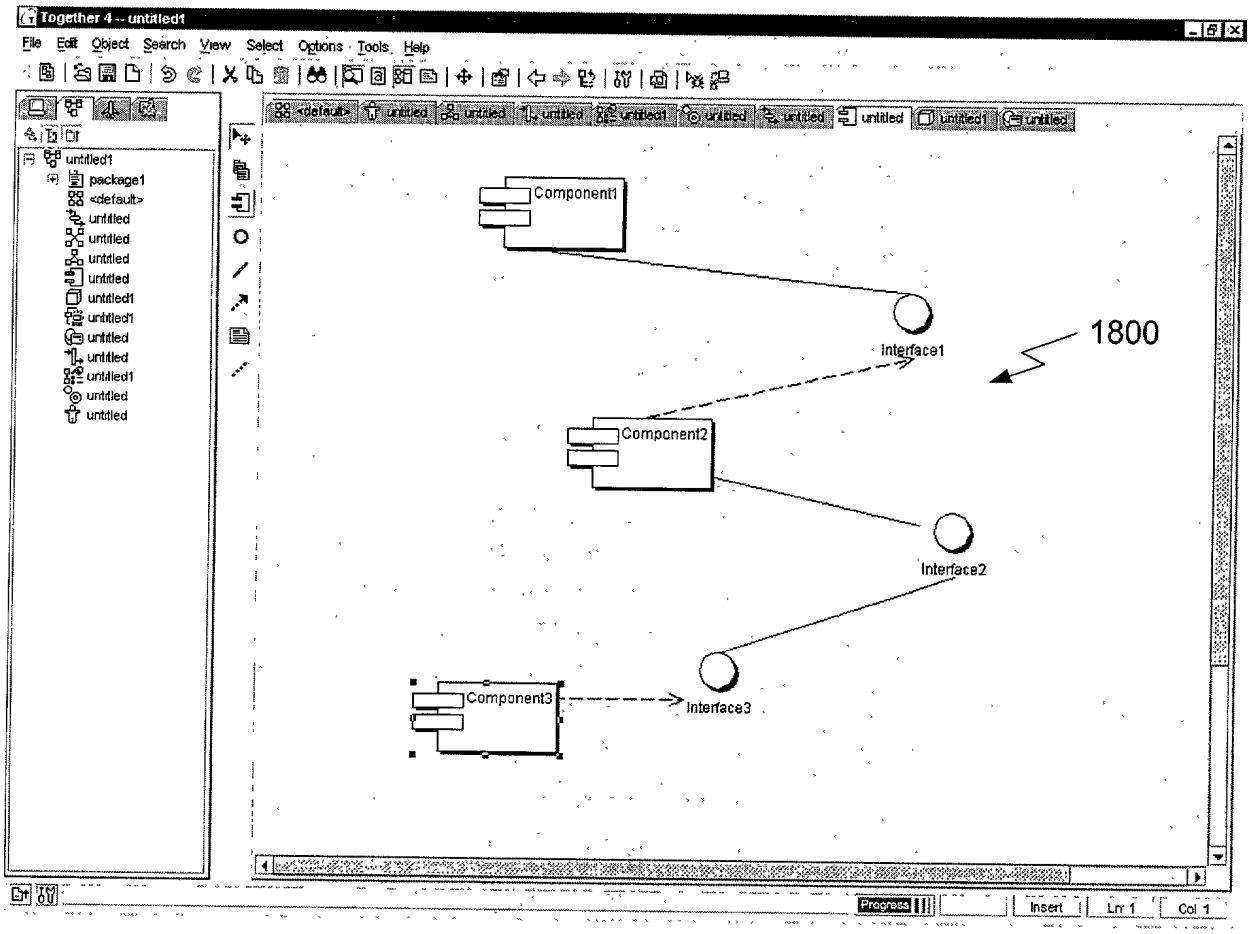
Together 4 -- untitled1
File Edit Object Search View Select Options Tools Help
untitled1
package1
<default>
Component1
Interface1
1800
Component2
Interface2
Component3
Interface3
Progress
Insert
Ln 1
Col 1

FIG. 18

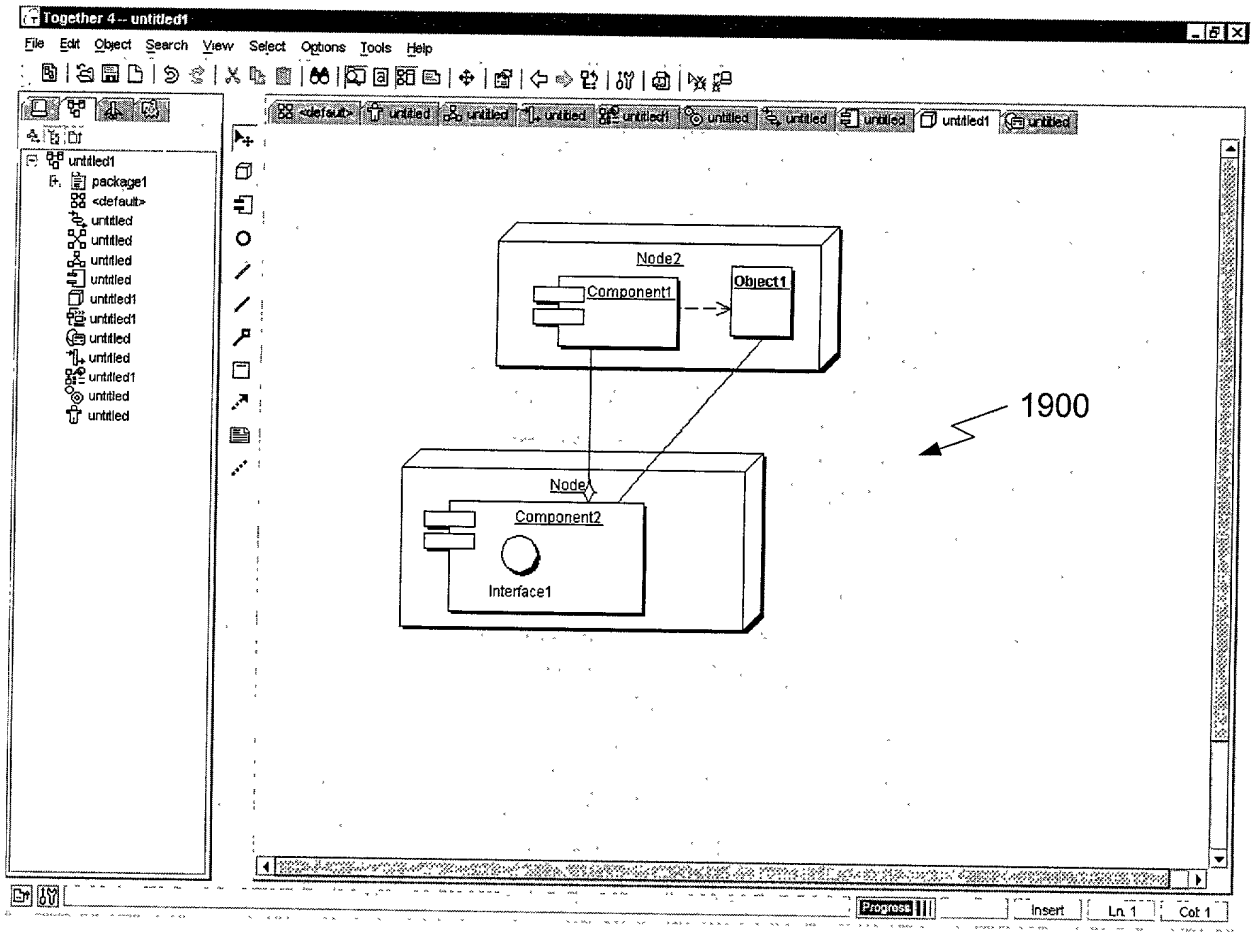
Together 4 -- untitled1
File Edit Object Search View Select Options Tools Help
untitled1
package1
<default>
Node2
Component1
Object1
1900
Node
Component2
Interface1
Progress
Insert
Ln 1
Col 1

FIG. 19

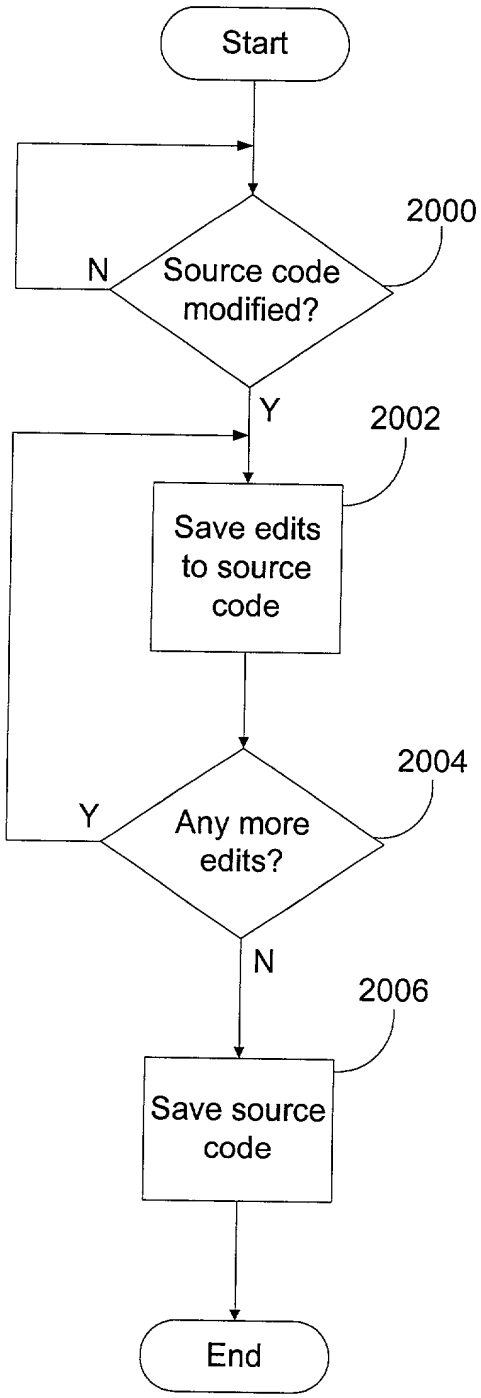
Start
2000
N
Source code modified?
Y
2002
Save edits to source code
2004
Y
Any more edits?
N
2006
Save source code
End

FIG. 20

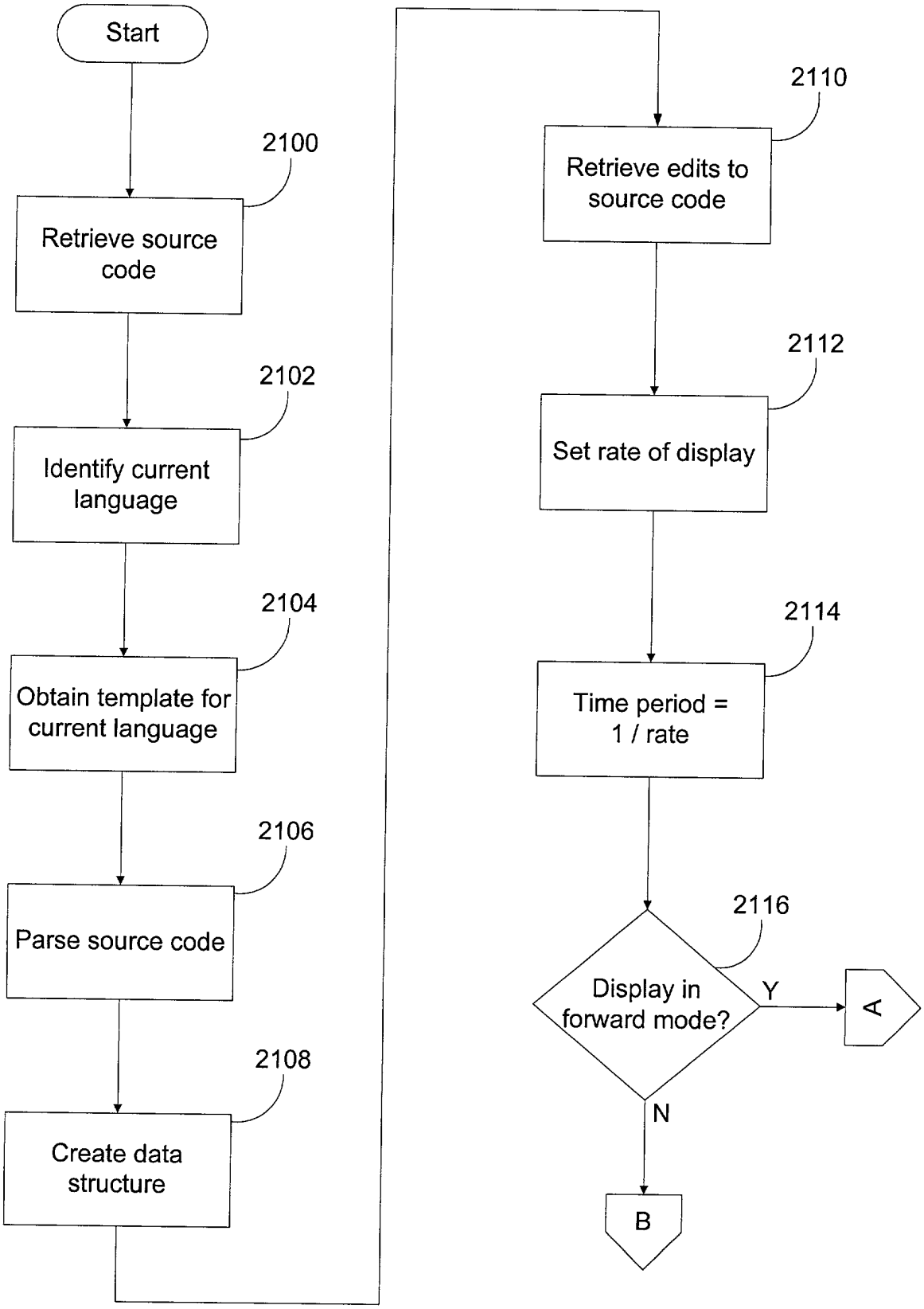
Start
2100
Retrieve source code
2102
Identify current language
2104
Obtain template for current language
2106
Parse source code
2108
Create data structure
2110
Retrieve edits to source code
2112
Set rate of display
2114
Time period = 1 / rate
2116
Display in forward mode?
Y
A
N
B

FIG. 21A

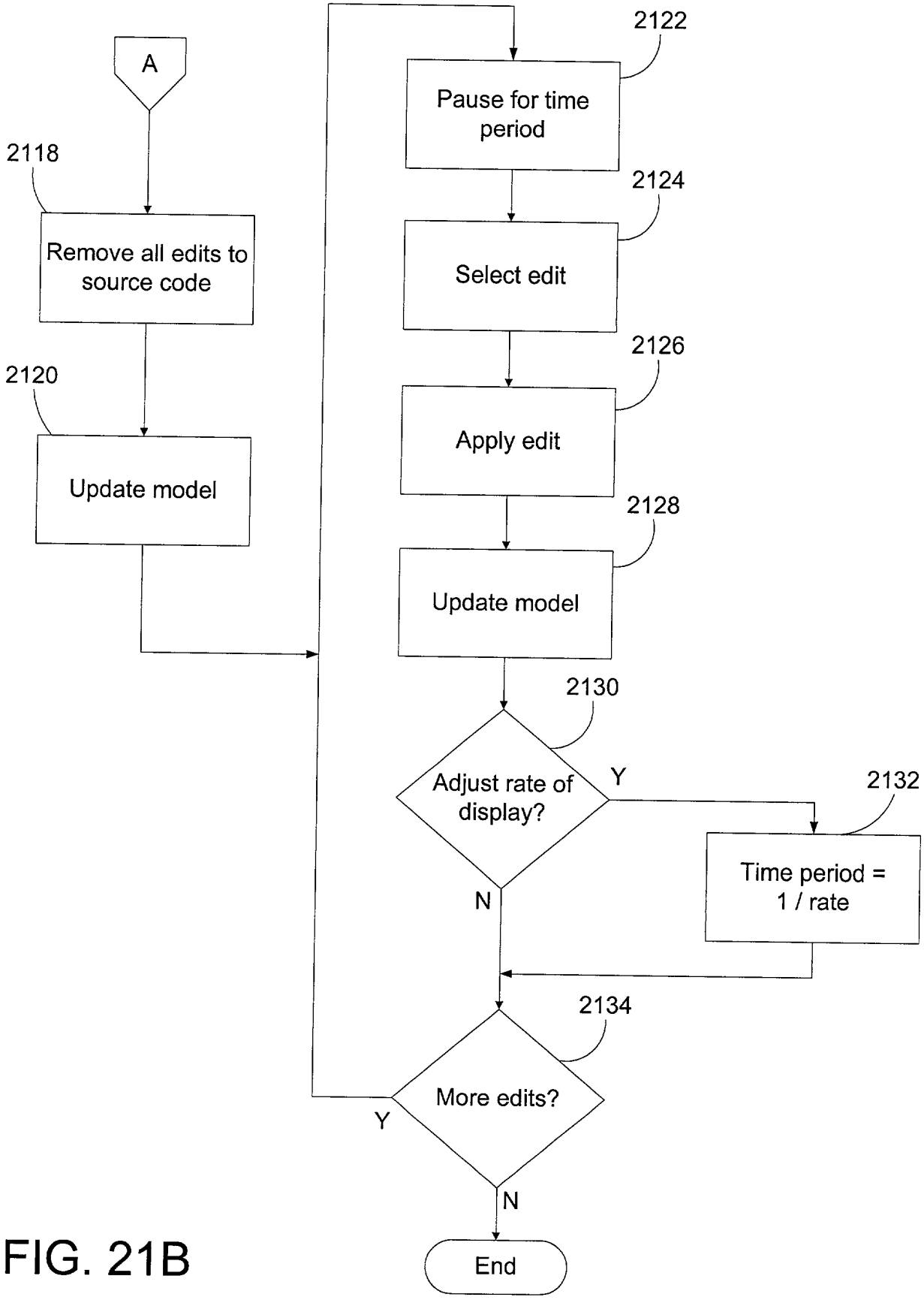
A
2118
Remove all edits to source code
2120
Update model
2122
Pause for time period
2124
Select edit
2126
Apply edit
2128
Update model
2130
Adjust rate of display?
Y
2132
Time period = 1 / rate
N
2134
More edits?
Y
N
End

FIG. 21B

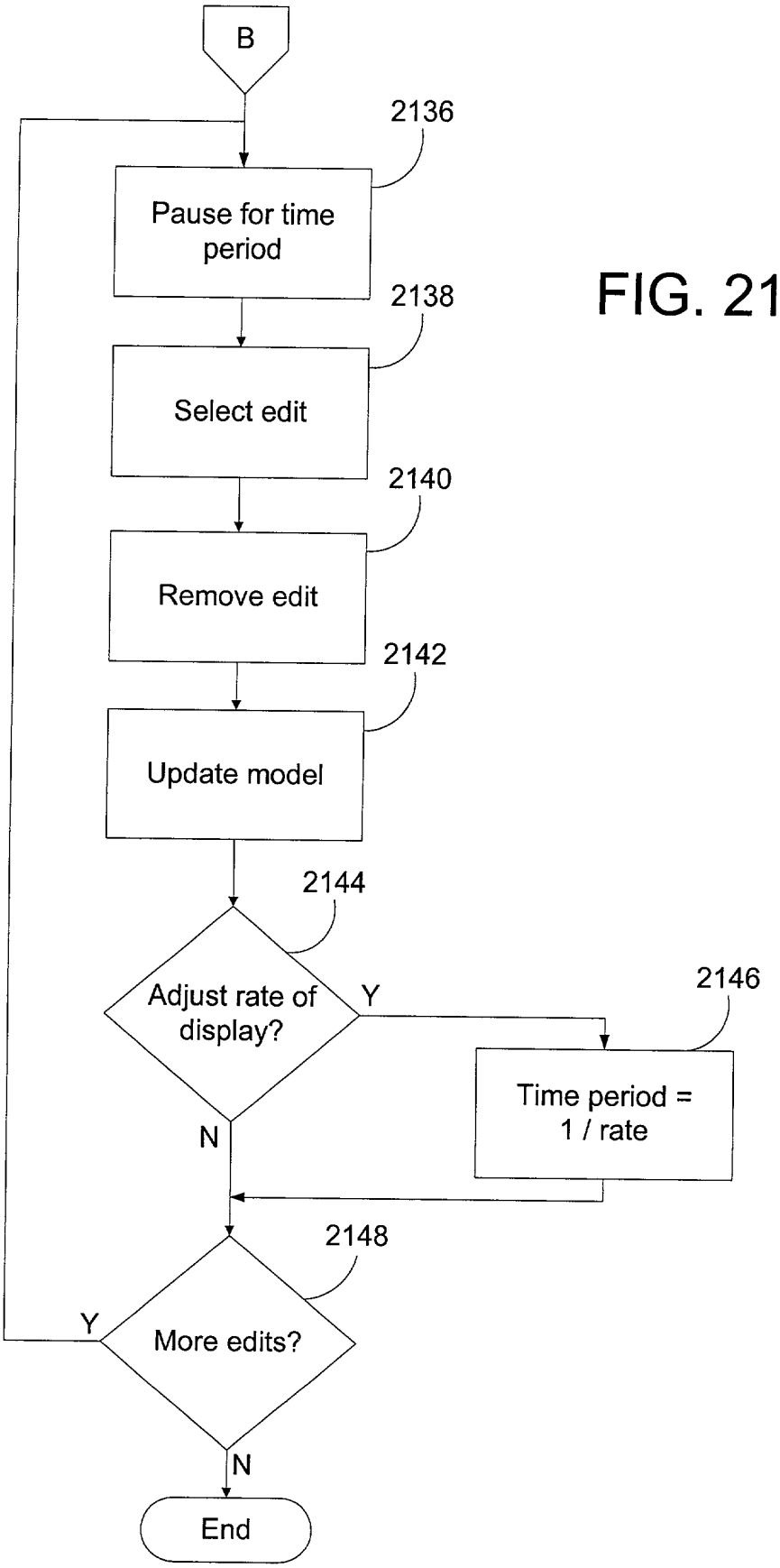
B
2136
Pause for time period
2138
Select edit
2140
Remove edit
2142
Update model
2144
Adjust rate of display?
Y
2146
Time period = 1 / rate
N
2148
More edits?
Y
N
End

FIG. 21C

DECLARATION AND POWER OF ATTORNEY

As a below named inventor, I hereby declare:

That my residence, post office address and citizenship are as stated below next to my name.

That I verily believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural inventors are named below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

METHOD AND SYSTEM FOR DISPLAYING CHANGES OF SOURCE CODE

the specification of which (check one)

(x) is attached hereto.

( ) was filed on _____ as
Application Serial No. _____
and was amended on _____

That I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above.

That I acknowledge the duty to disclose information known to be material to patentability of this application in accordance with Title 37, Code of Federal Regulations, §1.56(a).

That I hereby claim foreign priority benefits under Title 35, United States Code, §119 of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate on this invention having a filing date before that of the application on which priority is claimed:

Prior Foreign Application(s)			Priority Claimed	
_____	_____	_____	☐	☐
(Number)	(Country)	(Day/Month/Year Filed)		
_____	_____	_____	☐	☐
(Number)	(Country)	(Day/Month/Year Filed)	Yes	No

I hereby claim the benefit under 35 U.S.C. § 119(e) of any United States provisional application(s) listed below.

60/157,826	October 5, 1999
(Application Number)	(Filing Date)
60/199,046	April 21, 2000
(Application Number)	(Filing Date)

That I hereby claim the benefit under Title 35, United States Code, §120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, §112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, §1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

United States Application(s)

(Application Serial No.)	(Filing Date)	(Status)-(Patented, pending, abandoned)
(Application Serial No.)	(Filing Date)	(Status)-(Patented, pending, abandoned)

That all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issuing thereon.

I hereby appoint the following attorneys, with full power of substitution and revocation, to prosecute this application and to transact all business in the United States Patent and Trademark Office connected therewith and request that all correspondence and telephone calls in respect to this application be directed to Marina N. Saito, SONNENSCHEIN, NATH & ROSENTHAL, P. O. Box 061080; Wacker Drive Station, Sears Tower, Chicago, Illinois 60606-1080

Attorney	Registration No.	Attorney	Registration No.
Howard B. Rockman	22,190	Marina N. Saito	42,121
Kevin W.Guynn	29,927	Lana M. Knedlik	42,748
David R. Metzger	32,919	Alison P. Schwartz	43,863
Janelle D. Strode	34,738	Gregory B. Gulliver	44,138
Michael L. Kiklis	38,939	Christopher P. Rauch	45,034
Joseph A. Mahoney	38,956	Francisco A. Rubio-Campos	45,358
Jordan A. Sigale	39,028	Brian J. Gill	P46,727
Michael A. Molano	39,777	Shashank S. Upadhye	N/A – ltd. authority
Jennifer H. Hammond	41,814		

Full name of sole or one joint inventor: Peter Coad

Inventor's signature: _____

Date: _____

Residence and Post Office Address: 1720 Leigh Drive
Raleigh, North Carolina 27603

Citizenship: US

Address for Correspondence: Ms. Marina N. Saito
Sonnenschein Nath & Rosenthal
P. O. Box 061080
Wacker Drive Station
Sears Tower
Chicago, IL 60606-1080

Full name of sole or one joint inventor	Dietrich Charisius
Inventor's Signature	_____
Date	_____
Residence and Post Office Address	Gablenbergerweg 26 70186 Stuttgart Germany
Citizenship:	Germany

0500005:1:HOU